STRUCTURES SUBJECT TO SPACE COMPLEXITY

By

ZIA UDDIN

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

2004

ACKNOWLEDGMENTS

I owe an immense debt of gratitude to my supervisory committee chair, Professor Douglas Cenzer. Without his guidance, this work simply would not have existed. He has been unfailingly generous to me, patient, encouraging, sympathetic, and full of good humor. I am very lucky that he introduced me to the subject of Complexity Theory, and I am sure I am lucky in more ways than I realize to have him as an advisor.

I would also like to thank the other members of my supervisory committee: Dr. William Mitchell, Dr. Beverly Sanders, Dr. Rick Smith, and Dr. Helmut Voelklein. I have learned a lot from them either through taking classes taught by them, personal conversations, or hearing them speak in seminars and conferences.

I am very grateful to the Department of Mathematics for supporting me with a Teaching Assistantship during my years as a graduate student. The staff and many of the professors have been very generous with their time and help over the years.

Finally, I must acknowledge the contribution of my parents, my brother, all my good friends, and other sources that I may not be aware of, toward my progress along this path.

TABLE OF CONTENTS

Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

STRUCTURES SUBJECT TO SPACE COMPLEXITY

By

Zia Uddin

August 2004

Chair: Douglas A. Cenzer
Major Department: Mathematics

Complexity Theory has been applied to Model Theory and Algebra by Nerode, Cenzer, Remmel, and others. In their studies, the primary focus has been on polynomial-time complexity and other notions of bounded time. We now examine the notions of bounded space complexity in Algebra and Model theory. Of particular interest are the classes of logarithmic-space, linear-space, and polynomial-space computable sets and functions.

CHAPTER 1
INTRODUCTION

1.1 Basic Themes in Complexity-Theoretic Mathematics

This work falls under the subject of Complexity-Theoretic Model Theory and Algebra. In this subject, one uses the rich theory of Recursive Model Theory and Recursive Algebra as a reference but looks at resource (e.g., time or space) bounded versions of the results in those areas. Recursive Model Theory and Algebra deals with effective (in other words, recursive, i.e., computable) versions of results from ordinary Model Theory and Algebra. As an example, let us consider the following fact from Algebra: *Every linearly independent subset of a vector space can be extended to a basis.* All the known proofs of this fact for infinite-dimensional vector spaces use the Axiom of Choice and, therefore, are nonconstructive. As a result, one might conjecture that there is a linearly independent subset of a vector space that cannot be extended to a *recursive* basis. This conjecture was in fact proved by Metakides and Nerode [14] in 1975. Hence, in this particular case, the analogous recursive-algebraic statement of a theorem of Algebra turns out to be false. As an example of a positive result, Dekker [9] proved that every recursively-presented infinite-dimensional vector space over a recursive field with a dependence algorithm has a recursive basis. Here, a dependence algorithm is one that can determine uniformly effectively whether any set of n vectors $v_0, \ldots, v_{n-1}$ are dependent.

In the two decades before the work of Metakides and Nerode, there were vast developments in many areas of theoretical computer science (Hopcroft and Ullman [12]), particularly in complexity theory (Hartmanis and Stearns [11]). In due course, Nerode and Remmel [15-20] began the investigation of complexity-theoretic analogues of results from Recursive Algebra and other areas of mathematics. The natural complexity-theoretic analogue of Dekker's result is that every polynomial-time infinite-dimensional vector space over a polynomial-time field with a polynomial-time dependence algorithm has a polynomial-time basis. Nerode and Remmel [17] proved this statement only in the case where the underlying field is infinite and has a polynomial-time representation with certain nice properties. If, however, the underlying field is finite and we do not have a dependence algorithm, then this statement turns out to be oracle dependent. Nerode and Remmel [17] also gave a general construction which was used afterwards by Cenzer and Remmel [8; Theorem 8.14] to prove that there exists an infinite-dimensional vector space without any polynomial-time basis.

In general, the natural complexity-theoretic analogue of a result in Recursive Model Theory and Algebra may be false because the proof of the recursive result uses the unbounded resources allowed in recursive constructions in a crucial way. In contrast, there are results in Recursive Model Theory and Algebra for which the natural complexity-theoretic analogue is true but requires a more delicate proof that incorporates the resource bounds. Furthermore, a number of new and interesting phenomena arise in complexity-theoretic investigations because of two facts.

First, whereas all infinite isomorphic recursive sets are recursively isomorphic, not all infinite polynomial-time sets are polynomial-time isomorphic. In particular, Tal(ω), the tally representation of the set of natural numbers ω, is not polynomial-time isomorphic to Bin(ω), the binary representation of ω. Hence, in Complexity-Theoretic Model Theory and Algebra, it does make a difference if we choose Tal(ω) as our universe instead of Bin(ω). Of course, in Recursive Model Theory and Algebra, these two representations of ω are interchangeable.

Second, complexity-theoretic results do not relativize as is the case for most recursion-theoretic results. For example, Baker, Gill, and Solovay [1] proved that there are recursive oracles X and Y such that $P^X = NP^X$ and $P^Y \neq NP^Y$. We also recall the oracle-dependency of one part of the complexity-theoretic analogue of Dekker's result above. In Recursive Model Theory and Algebra, it has generally been the case that if two classes of recursive structures are equal, then they are equal relative to all oracles.

In the remainder of this section we will briefly describe the themes of and survey some results in Complexity-Theoretic Model Theory and Algebra. Section 1.2 discusses how our present work relates to previous research in this area, and then provides an outline of this work. Precise definitions are given in Section 1.3.

There are basically two major themes in Complexity-Theoretic Model Theory and Algebra. The first theme is Complexity-Theoretic Model Theory, which deals with model existence questions. The second theme (Complexity-Theoretic Algebra) fixes a given structure (e.g., a polynomial-time structure) and explores the properties of that structure. Our work is concerned with the first theme. This first theme (Complexity-Theoretic Model Theory) itself has two major subthemes. In the first subtheme, the focus so far has been on comparing various recursive structures with feasible structures, and also comparing those recursive structures with feasible structures with a specified universe. Here, feasible is interpreted to be some relatively low time-complexity class such as P (polynomial-time) or EX (exponential-time). The second subtheme involves the problem of feasible categoricity and has been investigated by Cenzer and Remmel [4, 6, 7, 21]. Our work, however, is concerned mainly with the first subtheme.

Working within the first subtheme involves picking a complexity class $\mathcal{A}$ and asking which structures in some other class $\mathcal{C}$ can be represented by models in $\mathcal{A}$. By far, the complexity class that has received the most attention is P. In view of that, the four existence questions that have been posed for any class $\mathcal{C}$ of structures are as follows:

- Is every recursive structure in $\mathcal{C}$ isomorphic to some polynomial-time structure?
- Is every recursive structure in $\mathcal{C}$ *recursively* isomorphic to some polynomial-time structure?
- Is every recursive structure in $\mathcal{C}$ isomorphic to some polynomial-time structure with a *specified* universe such as the binary or tally representation of the natural numbers?
- Is every recursive structure in $\mathcal{C}$ *recursively* isomorphic to some polynomial-time structure with a *specified* universe such as the binary or tally representation of the natural numbers?

In contrast to these four existence questions pervading the first subtheme, the second subtheme of feasible categoricity involves asking the corresponding uniqueness questions. Returning to the existence questions, Grigorieff [10] studied the class $\mathcal{C}$ of linear orderings. He showed that every recursive linear ordering is *recursively* isomorphic to a polynomial-time (in fact, a real linear-time) linear ordering, thus answering the first two existence questions in the affirmative. To answer the third and fourth existence questions for the class of linear orderings, one must specify the universe of the alleged polynomial-time linear ordering in advance. We recall that the choice of any particular universe as opposed to another may well make a difference. Grigorieff showed that every recursive linear ordering is isomorphic to a polynomial-time linear ordering (again, a real linear-time ordering, in fact) whose universe is $\mathrm{Bin}(\omega)$, thus answering the third existence question in the affirmative for the particular universe $\mathrm{Bin}(\omega)$. However, Cenzer and Remmel [2] answered the fourth existence question for this particular universe in the negative by constructing a recursive linear ordering that is not *recursively* isomorphic to any polynomial-time linear ordering with universe $\mathrm{Bin}(\omega)$.

More generally, Grigorieff [10] showed that every recursive structure with a finite number of relation symbols and no function symbols is *recursively* isomorphic to a polynomial-time structure with a standard universe (i.e., whose universe can be taken to be the natural numbers in either their tally or their binary representation). Cenzer and Remmel [2] strengthened this result by showing that any purely relational recursive structure is *recursively* isomorphic to a polynomial-time structure with a standard universe. In contrast, it was also shown [2] that a recursive structure with a single unary function exists that is not *recursively* isomorphic to any polynomial-time structure; and that a recursive structure with one unary relation and one unary function exists that is not even isomorphic to any polynomial-time structure.

In addition to investigating whether recursive linear orderings, recursive relational structures, and recursive structures with unary functions have feasible models, researchers have investigated whether recursive Abelian groups [13], recursive vector spaces, recursive Boolean algebras, recursive graphs [6], and recursive permutation structures have feasible models. For example, two important positive results [2] are as follows:

- All recursive Boolean algebras are *recursively* isomorphic to polynomial-time structures.
- The standard model of arithmetic, namely, the structure $(\omega, S, +, \cdot, <, 2^x)$, is *recursively* isomorphic to a polynomial-time structure.

As for examples of both positive and negative results, it was shown [3; pp. 343-348] that any recursive torsion Abelian group G is *recursively* isomorphic to a polynomial-time group A; and that if the orders of the elements of G are bounded, then A may be taken to have a standard universe. It was also shown [3; p. 357] that a recursive torsion Abelian group exists that is not even isomorphic to any polynomial-time (or any primitive recursive) group with a standard universe.

1.2 Outline of Dissertation

The bounded resource in all of the investigations mentioned above is time. Our work involves carrying out some of the same investigations except that our bounded resource is space instead of time. For example, we are interested to know (following Cenzer and Remmel [2]) whether any purely relational recursive structure is *recursively* isomorphic to a logarithmic-space structure with a standard universe. Our focus is mainly on logarithmic space; and to a lesser extent on linear, polynomial, and exponential space. The importance of logarithmic space is a consequence of the following observations:

- Any logarithmic-space algorithm is a polynomial-time algorithm, whereas a linear-space algorithm is not necessarily a polynomial-time algorithm.
- Algorithms that can be carried out within polynomial time are generally thought to be the practically feasible ones.
- Algorithms that can be carried out within logarithmic space correspond to those that do not use up too much computer memory.

Chapter 2 begins by investigating the space complexity of the basic arithmetical operations. In the case of addition and subtraction, the investigation simply involves verifying that the standard algorithms for addition and subtraction belong to a low space-complexity class (Lemmas 2.1.1–2.1.7). The detailed verifications of the algorithms, although well-known, serve to elucidate later proofs. However, in the case of binary multiplication, we are obliged to give a modification of the standard algorithm (Lemma 2.1.8) since the standard algorithm requires linear space instead of logarithmic space. And thus far, we have been unsuccessful in arriving at a division algorithm that requires at most logarithmic space. (We note that both the standard binary multiplication and division algorithms are polynomial-time.) Hence, to prove the existence of a logarithmic-space bijection between the set of natural numbers written in binary and the same set written in base 3 or a higher base, we need to first prove the existence of such bijections between the natural numbers in binary and various other sets, which in turn are proved to have logarithmic-space bijections with the natural numbers in base 3 or higher. We then compose these bijections to obtain the desired result.

Accordingly, Section 2.2 is a necessary detour where we investigate the space complexity of function composition. Once again, we begin by reproving well-known results (Lemmas 2.2.1–2.2.3) since we need to refer to their proofs, and not just the statements, later on. We then go on to prove a series of space composition lemmas

(Lemmas 2.2.3–2.2.9) some of which deal with the situations where a space complexity class is closed under function composition. We end the section by proving the Generalized Space Composition Lemma (Lemma 2.2.12), which is then used throughout our work.

Section 2.3 deals with various applications of the space complexity of function composition. For example, we construct a logarithmic-space bijection from $\mathbb{N} \times \mathbb{N}$ to $\mathbb{N}$ (Lemma 2.3.3), the standard bijection (Lemma 2.3.2) being a linear-space bijection, and not provably a logarithmic-space bijection. Lemmas 2.3.4–2.3.9 all serve to eventually prove the existence of a logarithmic-space bijection between the set of natural numbers written in binary and the same set written in a higher base (Lemma 2.3.10). Later, we prove Lemma 2.3.13, which characterizes those LOGSPACE subsets of $\mathrm{Tal}(\omega)$ that are in fact LOGSPACE set-isomorphic to the whole of $\mathrm{Tal}(\omega)$; and Lemma 2.3.14, which deals with Cartesian products and disjoint unions, and which proves to be crucial in constructing structures (models) throughout Chapter 3.

Section 3.1 includes the basic model-building lemmas. Later, we rely particularly on Lemma 3.1.1 and Lemma 3.1.3 (b). Lemma 3.1.1 deals with the most basic situation where the LOGSPACE complexity of a structure is preserved under the action of a LOGSPACE bijection. It so happens that LOGSPACE complexity is, in general, not preserved when we move from the "binary version" of a structure to the "tally version." The most important exception from our point of view is dealt with in Lemma 3.1.3 (b).

Section 3.2 begins by answering in the affirmative (Theorem 3.2.1) the question posed earlier, namely, whether any purely relational recursive structure is *recursively* isomorphic to a logarithmic-space structure with a standard universe. Next, we show (Theorem 3.2.2) that an affirmative answer is impossible in the case of purely functional recursive structures. In the proof of Theorem 3.2.2, we in fact end up constructing a permutation structure that has both finite and infinite orbits. Hence, we reach a natural spot in our work where we can begin to investigate the space complexity of permutation structures according to the number and size of their orbits. Theorem 3.2.4 shows that we cannot specify a standard universe even for a recursive permutation structure where all the orbits are finite. However, we can successfully specify a standard universe by putting further restrictions on the orbits, as in the cases where we have only a finite number of orbits (Theorem 3.2.5); at least one but only a finite number of infinite orbits (Theorem 3.2.6); all obits of the same finite size (Theorem 3.2.7); and an upper bound on the size of the orbits (Corollary 3.2.9). In contrast, for finitary permutation structures that do not have an infinite number of orbits of the same fixed size, we put restrictions on their spectrum (defined in Section 1.3). In this way, we obtain both positive results [Theorems 3.2.12 and 3.2.14, where the spectrum is a LOGSPACE subset of Tal(ω); and Corollary 3.2.13] where we can specify a standard universe, and a negative result (Theorem 3.2.15). Our final result for permutation structures is a negative one (Theorem 3.2.16) where we quote two examples with an infinitely number of infinite orbits that are not recursively isomorphic to any LOGSPACE structure.

Section 3.3 deals with the space complexity of Abelian groups. Results here parallel those for permutation structures. We define the direct product of a sequence of groups, and consider the space complexity of this product in Lemma 3.3.2. This lemma is a foundation for proving results about the basic groups: In Lemma 3.3.3, we prove that the basic groups (like $\mathbb{Z}$ and $\mathbb{Q}$) have standard LOGSPACE representations. In Lemma 3.3.4 we prove the same about finite products of the basic groups. Then in Theorem 3.3.5, we prove the same once again for finitely-generated Abelian groups. Next, we consider recursive torsion Abelian groups, which is where we have similarities with the results for permutation structures. For example, Theorem 3.3.8 (where the "orbit" of every group element is finite) is the analogue of Theorem 3.2.4; while Theorem 3.3.13 (dealing with torsion Abelian groups with an upper bound on the orders of the elements) is the analogue of Corollary 3.2.9. We conclude our work by considering torsion Abelian groups with no upper bound on the orders of the elements. Once again, we define and put restrictions on the spectrum of torsion Abelian groups. This time, the positive result of Theorem 3.3.17 is the analogue of Theorem 3.2.14. And we have negative results in this case as well. The remainder of Chapter 1 gives the necessary definitions and establishes notation.

1.3 Definitions

Let Σ be a finite alphabet. Then Σ^* denotes the set of finite strings of letters from Σ, and Σ^ω denotes the set of infinite strings of letters from Σ, where $\omega = \{0, 1, 2, \dots\}$ is the set of natural numbers. For a symbol $s \in \Sigma$ and for each natural number $n \neq 0$, s^n denotes the string of n of the symbols s, while s^0 denotes the empty string $\emptyset$.

For a string $\sigma = (\sigma(0), \sigma(1), \dots, \sigma(n-1))$, where $\sigma(i-1)$ is the ith symbol of σ, $1 \leqslant i \leqslant n$, the symbol $|\sigma|$ denotes the length n of σ. The empty string $\emptyset$ has length 0. For $m < |\sigma|$, the string $(\sigma(0), \dots, \sigma(m-1))$ is denoted by $\sigma\lceil m$. A string σ is an *initial segment* of a string τ, written $\sigma \prec \tau$, if $\sigma = \tau\lceil m$ for some $m < |\tau|$. The *concatenation* $\sigma^\frown\tau$ (or sometimes just $\sigma\tau$) is defined by

$$\sigma^\frown\tau = (\sigma(0), \dots, \sigma(m-1), \tau(0), \dots, \tau(n-1)),$$

where $|\sigma| = m$ and $|\tau| = n$. In particular, we write $\sigma^\frown a$ for $\sigma^\frown(a)$ and $a^\frown\sigma$ for $(a)^\frown\sigma$, where (a) is the string consisting of the single symbol $a \in \Sigma$.

For any natural number $n \neq 0$, $\mathrm{tal}(n) = 1^n$ is the tally representation of n and $\mathrm{bin}(n) = i_0 i_1 \dots i_j \in \{0,1\}^*$ is the (reverse) binary representation of n if $n = i_0 + i_1 \cdot 2 + \dots + i_j \cdot 2^j$ and $i_j \neq 0$. And more generally for $k > 2$, the (reverse) k-ary representation of n is $\mathrm{b}_k(n) = i_0 i_1 \dots i_j \in \{0, 1, \dots, k-1\}^*$, if $n = i_0 + i_1 \cdot k + \dots + i_j \cdot k^j$ and $i_j \neq 0$. We let $\mathrm{tal}(0) = \mathrm{bin}(0) = \mathrm{b}_k(0) = 0$. Then we let $\mathrm{Tal}(\omega) = \{\mathrm{tal}(n) : n \in \omega\}$, $\mathrm{Bin}(\omega) = \{\mathrm{bin}(n) : n \in \omega\}$, and $\mathrm{B}_k(\omega) = \{\mathrm{b}_k(n) : n \in \omega\}$

for each $k \geqslant 3$. Occasionally, we will want to say that $B_2(\omega) = \mathrm{Bin}(\omega)$ and $B_1(\omega) = \mathrm{Tal}(\omega)$.

Given $A, B \subseteq \{0,1\}^*$, we let $A \times B = \{\langle a, b\rangle : a \in A, b \in B\}$, the ordinary Cartesian product. Let $A \oplus B = \{\langle 0, a\rangle : a \in A\} \cup \{\langle 1, b\rangle : b \in B\}$, which we call the *disjoint union of A and B*. More generally, given $A_1, \ldots, A_n \subseteq \{0, 1, \ldots, k-1\}^*$, we define the *$n$-fold disjoint union* $A_1 \oplus A_2 \oplus \cdots \oplus A_n$ as $A_1 \oplus A_2 \oplus \cdots \oplus A_n = \{\langle 0, a_1\rangle : a_1 \in A_1\} \cup \{\langle 1, a_2\rangle : a_2 \in A_2\} \cup \cdots \cup \{\langle n-1, a_n\rangle : a_n \in A_n\}$.

Our model of computation is the multitape Turing machine of Papadimitriou [23]. We recall that in this model, the alphabet Σ of a machine always contains the blank symbol $\sqcup$ and the first symbol $\triangleright$. If $x \in (\Sigma \setminus \{\sqcup, \triangleright\})^*$ is the current nonempty string on any tape, then we imagine the contents of that tape to be $\triangleright x \sqcup \sqcup \sqcup \ldots$ The definition of a machine's program ensures that $\triangleright$ is never overwritten and that upon reading $\triangleright$ on any tape, the cursor of that tape moves right. Also strings never become shorter in this model because no nonblank symbol is ever overwritten by the blank symbol $\sqcup$. Another important feature of this model is that the cursor of each tape can move independently of the cursors of other tapes.

Unless stated otherwise, our machines are both *read-only* (input strings are never overwritten) and *write-only* (the output-string cursor never moves left; and moves right immediately after each output symbol is written). Papadimitriou refers to such machines as *machines with input and output.*

A function $g(x)$ is a *proper complexity function* if $g(x)$ is identically a constant or one of $k \log x$, $(\log x)^k$, kx, x^k, $2^{(\log x)^k}$, 2^{kx}, 2^{x^k}, and $2^{2^{kx}}$, where k is a nonzero constant. For us $\log x$ is an abbreviation for $\log_2 x$.

Let $g : \omega \to \omega$ be a proper complexity function. A (deterministic) Turing machine M is said to be *g time bounded* if each computation of M of inputs of length $l \geqslant 2$ requires at most $g(l)$ steps. (If M has more than one input string, then l is taken to be the sum of the lengths of the input strings.) A (deterministic) read-only write-only Turing machine M is said to be g *space bounded* if at the end of each computation of M of inputs of length $l \geqslant 2$, the maximum length of a string on the work tapes (i.e., the non-input and non-output tapes) is $g(l)$. However, if M is not read-only or write-only, then we take into account the maximum lengths of strings on the input tapes and the output tape if they get written on, as well as the lengths of strings on the work tapes.

Once again, let $g : \omega \to \omega$ be a proper complexity function. A function $f : \omega^n \to \omega$ of $n \geqslant 1$ variables is said to be in $TIME(g)$ if there is a g *time bounded* Turing machine with n input tapes that computes f. The function f is said to be in $SPACE(g)$ if there is a g *space bounded* Turing machine with n input tapes that computes f. A set or relation is in $TIME(g)$ (resp. $SPACE(g)$) if its characteristic function is in $TIME(g)$ (resp. $SPACE(g)$).

We now define the important (deterministic) time complexity classes that we consider:

$$R = \bigcup_c \{TIME(x+c) : c \geq 0\},$$

$$LIN = \bigcup_c \{TIME(cx) : c \geq 0\},$$

$$P = \bigcup_c \{TIME(x^c) : c \geq 0\}$$

$$EX = \bigcup_c \{TIME(2^{cx}) : c \geq 0\},$$

$$EXP = \bigcup_c \{TIME(2^{x^c}) : c \geq 0\},$$

$$DOUBEX = \bigcup_c \{TIME(2^{2^{cx}}) : c \geq 0\}.$$

And if S is any of the above time complexity classes, then

$$DEX(S) = \bigcup_{g \in S} \{TIME(2^g)\}.$$

The smallest class Q containing P and closed under the DEX operator can be defined by iterating DEX so that $P^0 = P$, $P^{n+1} = DEX(P^n)$ for each n, and $Q = \bigcup_{n<\omega} P^n$.

Let 0 (resp. $k > 0$) denote the identically zero (repectively k) function. The (deterministic) space complexity classes that we consider are as follows:

$$\begin{aligned}
&ZEROSPACE \;=\; SPACE(0),\\
&CONSTANTSPACE \;=\; \bigcup_k\{SPACE(k) : k > 0\},\\
&LOGSPACE \;=\; \bigcup_c\{SPACE(c\log x) : c \geq 0\},\\
&PLOGSPACE \;=\; \bigcup_c\{SPACE((\log x)^c) : c \geq 0\},\\
&LINSPACE \;=\; \bigcup_c\{SPACE(cx) : c \geq 0\},\\
&PSPACE \;=\; \bigcup_c\{SPACE(x^c) : c \geq 0\},\\
&SUPERPSPACE \;=\; \bigcup_c\{SPACE(2^{(\log x)^c}) : c \geq 0\},\\
&EXSPACE \;=\; \bigcup_c\{SPACE(2^{cx}) : c \geq 0\},\\
&EXPSPACE \;=\; \bigcup_c\{SPACE(2^{x^c}) : c \geq 0\},\\
&EXSUPERPSPACE \;=\bigcup_c\{SPACE(2^{2^{(\log x)^c}} : c \geq 0\},\\
&DOUBEXSPACE \;=\; \bigcup_c\{SPACE(2^{2^{cx}}) : c \geq 0\},\\
&DOUBEXPSPACE \;=\; \bigcup_c\{SPACE(2^{2^{x^c}}) : c \geq 0\}.
\end{aligned}$$

We say that a function f is *quasi real-time* if $f \in R$. This is slightly more general than the usual notion of real time, since real-time functions are in $TIME(x)$. The function f is *linear-time* if $f \in LIN$; is *polynomial-time* or *p-time* if $f \in P$; is *exponential-time* if $f \in EX$; is *double exponential-time* if $f \in DOUBEX$; and is *q-time* if $f \in Q$.

Similarly, a function f is *zero-space* if $f \in ZEROSPACE$; is *constant-space* if $f \in CONSTANTSPACE$; is *logarithmic-space* if $f \in LOGSPACE$; is *polylogarithmic-space* if $f \in PLOGSPACE$; is *linear-space* if $f \in LINSPACE$; is *polynomial-space* or *p-space* if $f \in P$; is *exponential-space* if $f \in EXSPACE$; and is *double exponential-space* if $f \in DOUBEXSPACE$.

Odifreddi [22] includes all the basic definitions of recursion theory. Let $\phi_{i,n}$ be the partial recursive function of n variables computed by the ith Turing machine M_i. If $n = 1$, we write ϕ_i instead of $\phi_{i,1}$. Given a string $\sigma \in \Sigma$, we write $\phi_i^s(\sigma)\downarrow$ if M_i gives an output in s or fewer steps when started on input string σ. Thus the function ϕ_i^s is uniformly polynomial-time. We write $\phi_e(\sigma)\downarrow$ if $(\exists s)(\phi_e^s(\sigma)\downarrow)$, and $\phi_e(\sigma)\uparrow$ if it is not the case that $(\exists s)(\phi_e^s(\sigma)\downarrow)$. Throughout our work, we use the terms *recursive* and *computable* interchangably.

The structures we consider are structures over an effective language $\mathcal{L} = \langle \{R_i\}_{i\in S}, \{f_i\}_{i\in T}, \{c_i\}_{i\in U}\rangle$. Here S, T, and U are initial segments of ω, while c_i is a constant symbol for all $i \in U$. There are recursive functions s and t such that, for all $i \in S$, R_i is an $s(i)$-ary relation symbol; and, for all $i \in T$, f_i is a $t(i)$-ary function symbol. Also there are recursive functions σ and τ such that for all $i \in S$, $\sigma(i)$ is the index of a Turing machine that computes R_i; and, for all $i \in T$, $\tau(i)$ is the index of a Turing machine that computes f_i.

If a structure $\mathcal{A}$ has a universe (i.e., its underlying set) that is a subset of $\mathrm{Bin}(\omega)$, then by $\mathrm{tal}(\mathcal{A})$, we simply mean the structure isomorphic to $\mathcal{A}$ whose universe is a subset of $\mathrm{Tal}(\omega)$. We have a similar understanding of the notation $\mathrm{bin}(\mathcal{A})$ in the case where $\mathcal{A}$ has a universe that is a subset of $\mathrm{Tal}(\omega)$. A *relational structure* is a structure over a language that has no function symbols. A structure has *standard universe* if its universe is all of $\mathrm{Tal}(\omega)$ or $\mathrm{Bin}(\omega)$.

Let Γ be some class of sets (relations or functions) such as primitive recursive or partial recursive or some complexity class. We say that a set (relation or function) is Γ-*computable* if it is in Γ.

A structure $\mathcal{A} = \langle A, \{R_i^{\mathcal{A}}\}_{i\in S}, \{f_i^{\mathcal{A}}\}_{i\in T}, \{c_i^{\mathcal{A}}\}_{i\in U}\rangle$ (where the universe A of $\mathcal{A}$ is a subset of Σ^*) is a Γ-*structure* if the following conditions hold:

- A is a Γ-computable subset of Σ^*.
- For each $i \in S$, $R_i^{\mathcal{A}}$ is a Γ-computable $s(i)$-ary relation on $A^{s(i)}$. Or, more formally, $R_i^{\mathcal{A}}$ is the restriction to $A^{s(i)}$ of a Γ-computable relation R on $(\Sigma^*)^{s(i)}$.
- For each $i \in T$, $f_i^{\mathcal{A}}$ is a Γ-computable $t(i)$-ary function from $A^{t(i)}$ into A. Or, more formally, $f_i^{\mathcal{A}}$ is the restriction to $A^{t(i)}$ of a Γ-computable funcion f on from $(\Sigma^*)^{t(i)}$ into Σ^*.

For any pair of structures $\mathcal{A} = \langle A, \{R_i^{\mathcal{A}}\}_{i\in S}, \{f_i^{\mathcal{A}}\}_{i\in T}, \{c_i^{\mathcal{A}}\}_{i\in U}\rangle$ and $\mathcal{B} = \langle B, \{R_i^{\mathcal{B}}\}_{i\in S}, \{f_i^{\mathcal{B}}\}_{i\in T}, \{c_i^{\mathcal{B}}\}_{i\in U}\rangle$, we say that $\mathcal{A}$ and $\mathcal{B}$ are Γ-*isomorphic* if and only if there is an isomorphism f from $\mathcal{A}$ onto $\mathcal{B}$ and Γ-computable functions F and G such that $f = F|A$ (the restriction of F to A) and $f^{-1} = G|B$. We end this section by including some definitions related to permutations (i.e., bijections from a set onto itself) and groups.

Let f be an injection of a set A into itself, and let $a \in A$. The *orbit* $\mathcal{O}_f(a)$ of a under f is $\mathcal{O}_f(a) = \{b \in A : (\exists n \in \omega)(f^n(a) = b \vee f^n(b) = a)\}$. The *order* $|a|_f$ of a under f is the cardinality of $\mathcal{O}_f(a)$. A *permutation structure* (A, f) is a structure where A is a set and f is a permutation on A. Given an injection $f : A \rightarrow A$, we define the *spectrum* of f by $\mathrm{Spec}(A, f) = \{n \in \omega : \exists a \in A)(|a|_f = n)\}$, and the *finite* and *infinite parts* of A by $\mathrm{Fin}_f(A) = \{a \in A : |a|_f < \omega\}$ and $\mathrm{Inf}_f(A) = \{a \in A : |a|_f = \omega\}$. An injection $f : A \rightarrow A$ is called *finitary* if all the orbits of all elements of A under f are finite; and *monic* if there are no two disjoint orbits of the same size.

For a group, we will distinguish two types of computability. The structure of a group $\mathcal{G}$ (with universe G) is determined by the binary operation which we denote by the addition sign $+^G$, since we are interested in Abelian groups. We let e^G denote the additive identity of $\mathcal{G}$. The inverse operation, denoted by inv^G, may also be included as an inherent part of the group. Thus we have the following distinction: A group $\mathcal{G}$ is *Γ-computable* if $(G, +^G, e^G)$ is Γ-computable and is *fully Γ-computable* if $(G, +^G, inv^G, e^G)$ is Γ-computable.

CHAPTER 2
SPACE COMPLEXITY OF OPERATIONS

2.1 Arithmetical Operations

In this section, we describe in detail how Turing machines can carry out the basic arithmetical operations, and thus determine the space complexity classes to which these operations belong.

Lemma 2.1.1. *The standard algorithms for addition and subtraction of 1 in k-ary, where $k \geqslant 2$, can be carried out using* zero *space.*

Proof: Suppose we are given a k-ary number σ on the input tape of our machine. We recall that σ is written in reverse k-ary.

To add 1 to σ, if the first symbol s of σ is less than $k-1$, the machine writes $s+1$ on the output tape and then copies the rest of σ on to the output tape. But if the first symbol of σ is $k-1$, then the machine writes a 0 on the output tape. Then, each time the machine reads a $k-1$ on the input tape, it writes a 0 on the output tape, until, if at all, the machine reads the first symbol t of σ that is less than $k-1$. At this point, the machine writes $t+1$ on the output tape and then copies the rest of the input on to the output tape. However, if all the symbols of σ are $k-1$, then the machine writes a 1 on the output tape as soon as it reads the $\sqcup$ on the input tape. The input tape is read-only and the output tape is write-only, and hence no space is used in this computation.

To subtract 1 from σ, if the first symbol on the input tape is 0, the machine writes $k-1$ on the output tape. The machine continues to write $k-1$ on the output tape each time it reads a 0 on the input tape, until it reads the first nonzero symbol s on the input tape. At this point, the machine checks whether $s = 1$ and whether

s is the last symbol of σ. If so, the machine halts. Otherwise, the machine writes $s-1$ on the output tape, and then copies the rest of the input on to the output tape. Once again, the input tape is read-only and the output tape is write-only, and so no space is used.

□

We generalize Lemma 2.1.1 in Lemma 2.1.5, but the actual algorithms for the special cases of addition and subtraction of 1 will frequently prove useful. Next, we verify facts about the relative lengths of a natural number in the tally and in the k-ary representation, $k \geqslant 2$, and the space complexity of converting between tally and k-ary.

Lemma 2.1.2. *Let $\sigma = \mathrm{b}_k(n)$, where $k \geqslant 2$ and $n > 0$. We have*

(a) $k^{|\sigma|-1} < n+1 \leq k^{|\sigma|}$, *or, equivalently,* $|\sigma| < 1 + \log(n+1) \leq |\sigma| + 1$.

(b) *If $|\sigma| = t$ and $\tau = \mathrm{b}_k(t)$, then $|\tau| = O(\log|\sigma|)$.*

(c) *The computation $\mu_k(\mathrm{b}_k(n)) = 1^n$ can be carried out using* linear *space, while the inverse computation ${\mu_k}^{-1}(1^n) = \mathrm{b}_k(n)$ can be carried out using* logarithmic *space.*

Proof: (a) If all the symbols of σ are $k-1$, that is n is the largest k-ary number possible for its length $|\sigma|$, then $n = k^{|\sigma|} - 1$, and so $n + 1 = k^{|\sigma|}$. On the other hand, if n is the smallest k-ary number possible for its length $|\sigma|$, then the last symbol of σ is 1 and all the previous symbols are 0. In that case, we have $k^{|\sigma|-1} = n$ and so $k^{|\sigma|-1} < n+1$.

(b) By part (a), we have $|\tau| < 1 + \log(t+1) = O(\log t)$.

(c) To go from k-ary to tally, our machine first copies the k-ary number on the input tape on to a work tape. Then it keeps employing the standard algorithm to subtract 1 in k-ary on this worktape until the content of this tape becomes 0. Moreover, each time a 1 is subtracted from this work tape, the machine writes a 1 on the output tape. The content of the work tape will never be longer than the length of the original input. Hence, going from k-ary to tally uses up linear space.

To go from tally to k-ary, our machine first writes a 0 on a worktape. Then, each time the machine reads a 1 on the input tape, it adds 1 in k-ary on this work tape using the standard algorithm. As soon as the $\sqcup$ is encountered on the input tape, the machine copies the k-ary number on the work tape on to the output tape. By part (a), the length of the number on the work tape will be logarithmic in the length of the input. Hence, going from tally to k-ary uses up logarithmic space.

$\square$

We shall see later, as a consequence of Lemma 2.3.12, that *for each $k \geqslant 2$, there is no algorithm to convert a k-ary number to its tally form which uses up only logarithmic space.*

In the following lemmas, we adopt the usual convention that $m - n$ is set to 0 if $m < n$. We shall see in the next two lemmas that the usual arithmetic operations can be carried out without using up any space if we use the tally representation of the natural numbers.

Lemma 2.1.3. *In* Tal(ω), *the addition and subtraction of, and multiplication and division (with remainder) by a constant requires* zero *space.*

Proof: Let the constant be $c \in \text{Tal}(\omega)$, and suppose that the input is $x \in \text{Tal}(\omega)$. If $c = 0$, then our machine simply copies the input to the output tape in the cases of addition and subtraction, and writes a 0 on the output tape in the case of multiplication. None of these operations use up any space. Now assume $c \neq 0$. For each of the four operations, we will use four different machines, each having c special states, among others.

To add c to x, the machine first copies x on to the output tape if $x \neq 0$. Then the machine switches to the c states one after another, each time writing a 1 on the output tape.

To subtract c from x, the machine outputs 0 if $x = 0$. Otherwise, the machine switches to the first of the c states when it reads the first symbol of x. The machine then switches to the remaining $c - 1$ states, one after another, each time it reads a new symbol of x. If the machine encounters a $\sqcup$ on the input tape before or immediately after switching to the last of the c states, then we have $x \leqslant c$, and the machine outputs 0. But if the machine encounters a 1 on the input tape immediately after switching to the last of the c states, it copies this 1 and the remaining 1's of x on to the output tape.

To multiply x by c, the machine outputs 0 if $x = 0$. Otherwise, it switches to the c states, one after another, each time copying x on to the output tape.

To divide x by c, we use a machine with a set Q_1 of $c - 1$ states, and another disjoint set Q_2 of c states, among others. If $x = 0$, the machine outputs $0 \sqcup 0$, signifying that the quotient and the remainder, respectively, are both 0. Otherwise, it switches to the $c - 1$ states in Q_1, one after another, each time it reads a symbol of x. If the machine encounters the $\sqcup$ on the input tape while in one of the states in Q_1, the quotient is 0, while x is the remainder. So the machine writes $0\sqcup$ on the output tape, then copies x on to the output tape, and then halts.

If the machine exhausts the states in Q_1 without encountering the input tape's $\sqcup$, the quotient is nonzero. In that case, the machine resets the input cursor to its extreme left position (thus reading the input tape's $\triangleright$). Then it switches to the c states in Q_2, one after another, each time it reads a symbol of x. If the machine reads a 1 of x after switching to the last of the c states in Q_2, it writes a 1 on the output tape (the first symbol of the nonzero quotient), and then switches back to the first of the c states in Q_2. The process is then repeated. Now if at any point the machine encounters a $\sqcup$ on the input tape before switching to the last of the c states in Q_2, then we have a nonzero remainder. The machine writes a $\sqcup$ on the output tape (the 1's making up the quotient having been written before this $\sqcup$) followed by a 1, then switches back to the *previous* one of the c states, and then keeps writing a 1 on the output tape each time it switches back to a previous state until it switches back to the first of the c states in Q_2. Note that since the machine is reading a $\sqcup$ now, we have no contradiction with the machine's activity while in any of the c states in Q_2 earlier and reading a 1. The output tape now contains a string of 1's (the quotient), followed by a $\sqcup$, followed by a string of 1's (the remainder). However, if the machine encounters a $\sqcup$ on the input tape immediately after switching to the last of the c states in Q_2, then we have zero remainder. In that case, the machine simply writes a $\sqcup$ on the output tape followed by a 0, the 1's or 1 making up the quotient having already been written.

$\square$

Lemma 2.1.4. *The addition, subtraction, multiplication, and division (with remainder) functions from* $\mathrm{Tal}(\omega) \times \mathrm{Tal}(\omega)$ *to* $\mathrm{Tal}(\omega)$*, the length function from* $\mathrm{Tal}(\omega)$ *to* $\mathrm{Tal}(\omega)$*, and the order relation on* $\mathrm{Tal}(\omega)$ *are all in ZEROSPACE.*

Proof: Suppose we are given $x \sqcup y$ in tally on the input tape. To compute $x + y$ in tally, our machine copies x on to the output tape (provided $x \neq 0$), ignores the $\sqcup$ after the x on the input tape, and then copies y on to the output tape (provided $y \neq 0$). But if $x = y = 0$, then the machine outputs 0.

Given x on the input tape, to compute the length $|x|$ of x in tally, our machine simply copies x on to the output tape unless $x = 0$, in which case it outputs 1.

For the rest of the proof, we shall use Turing machines with two input tapes, with x on the first input tape and y on the second.

To compute $x - y$, the two input cursors advance simultaneously to the right. If $x = 0$, the machine outputs 0 (since, by convention, we let $x - y = 0$ if $x \leqslant y$), and if $y = 0$, the machine outputs x. If the $\sqcup$ is encountered on the first input tape either before or at the same time as the one on the scond input tape, then we have $x \leqslant y$, and the machine outputs a 0. Otherwise, as soon as the machine encounters the $\sqcup$ on the second input tape, it copies the rest of x (including the currently-read 1 of x) on to the output tape.

To compute $x \cdot y$, the machine outputs 0 if either x or y is 0. Otherwise, it copies x on to the output tape each time it reads a 1 of y.

To compute $x \div y$, if $x = 0$, the output is $0 \sqcup 0$, signifying that the quotient and the remainder, respectively, are both 0. Otherwise, the two input cursors advance simultaneously to the right. We have three possibilities: (i) If the $\sqcup$ is encountered on the first input tape before the one on the second, we have $x < y$, and the machine outputs $0 \sqcup x$. (ii) If $\sqcup$s are encountered on the input tapes at the same time, we have $x = y$, and the machine outputs $1 \sqcup 0$. (iii) If a $\sqcup$ is encountered on the second input tape first, the machine switches to a new state q, outputs a 1 (the first symbol of the quotient), the second input cursor moves left all the way back to the beginning of y (while the first input cursor stays fixed), and then the two input cursors again move simultaneously to the right. Now in the case of possibility (i), the machine outputs a $\sqcup$, the 1's making up the quotient having been already written, and then outputs a 1 each time the second input cursor moves left (thus writing the remainder) until the second input cursor cannot move left anymore. In the case of possibility (ii), the machine outputs a 1, followed by a $\sqcup$, followed by a 0. And in the case of possibility (iii), the machine switches to a new state q^*, outputs a 1, the second input cursor moves left all the the way back to the beginning of y (while the first input cursor stays fixed), the machine switches back to state q, and the procedure is then repeated. Thus division in tally does not use up any space.

Finally, to check whether $x \leqslant y$, the machine simply checks that of the two simultaneously advancing input cursors, the first one encounters a $\sqcup$ either before or at the same time as the second input cursor.

$\square$

In the next four lemmas, we shall see that, in contrast to the situation obtained on using the tally representation of the natural numbers, not all arithmetic operations can be carried out using zero space when we use a k-ary representation where $k \geqslant 2$.

For example, the standard k-ary multiplication and division algorithms use up *linear* space. We shall give a modification of the standard multiplication algorithm that uses up only logarithmic space. However, we begin with those k-ary operations that can in fact be carried out without using up any space.

Lemma 2.1.5. *For each $k \geqslant 2$, the addition and subtraction of a constant in* $\mathrm{B}_k(\omega)$ *requires* zero *space.*

Proof: Let the (nonzero) constant be $c \in \mathrm{B}_k(\omega)$, and suppose the input is $x \in \mathrm{B}_k(\omega)$.

To add c to x, we employ a machine with two special "carry" states, among others, which inform the machine whether or not to carry a 1. For each of the k possible symbols from $\{0, 1, \ldots, k-1\}$ currently read, the machine's program has instructions as to which of the k symbols to write on the output tape, and whether to carry a 1 or a 0. For example, suppose $c = 24$ in $\mathrm{B}_{10}(\omega)$. (We recall that in reverse 10-ary, c is of course 42.) Then two of the many instructions will be as follows: (i) If at the starting state you read a 9 on the input tape, then write a 3 on the output tape, move right on the output tape, switch to the "carry a 1" state, and move right on the input tape. (ii) If at the "carry a 1" state you read a 5 on the input tape, then write an 8 on the output tape, move right on the output tape, switch to the "carry a 0" state, and move right on the input tape. Hence if, say, $x = 1759$ (which on the input tape will be written as 9571), then the machine will write the correct units and tens digits of $1759 + 24$ on the output tape and then will not carry a 1 to the next symbol 7 on the input tape. Another necessary feature of the machine is that $\sqcup$'s on the input tape are treated as 0's for as long as necessary in case $|c| \geqslant |x|$. Evidently this machine uses up no space in its operation.

To subtract c from x, we employ a machine similar to the machine for addition. Once again, the instructions of the machine's program exhaustively specify which symbol to write on the output tape depending on the symbol currently read on the input tape. However, instead of two "carry" states, the machine is equipped with two "borrow" states that allow it to borrow a 1 or a 0 from the next symbol on the input tape rather than carry a 1 or a 0 to the next symbol.

$\square$

Lemma 2.1.6. *For each $k \geqslant 2$, the addition and subtraction functions from* $\mathrm{B}_k(\omega) \times \mathrm{B}_k(\omega)$ *to* $\mathrm{B}_k(\omega)$ *and the order relation on* $\mathrm{B}_k(\omega)$ *are all in ZEROSPACE.*

Proof: Suppose we are given x and y in k-ary on the first and second input tapes, respectively.

To compute $x + y$, we use a machine similar to the one in the proof of the previous lemma. Once again, the instructions for the machine exhaustively specify the symbol to write on the output tape and the "carry" state to switch to, depending on the symbol of x and of y currently read. The two input cursors move right simultaneously and by one place each time. One of the many machine instructions in the case of $k = 10$ will be as follows: If at the starting state you read a 6 on the first input tape and an 8 on the second input tape, then write a 4 on the output tape, move right on the output tape, switch to the "carry a 1" state, and move right on both input tapes. As a result, this machine will output the correct units digit of, for example, $1066 + 1918$ and carry a 1 to the tens digits of $x = 1066$ and $y = 1918$ (written as 6601 and 8191, respectively).

To compute $x - y$, the machine used is similar to the one for addition, except that a 1 or a 0 is borrowed from the next symbol of x rather than carried to the next symbol.

Finally, to check whether $x \leqslant y$, our machine just needs to check whether $x - y = 0$. Hence, this machine's program is the same as that of the machine for subtraction in the previous paragraph with the modification that the output $x - y$ is never written. Instead, the machine simply uses states and the explicit instructions to borrow 1 or 0 as appropriate until, if ever, one output symbol "would have been" nonzero had the original machine for subtraction been used. In this case, the machine outputs "no." Otherwise, the machine outputs "yes."

□

Lemma 2.1.7. *For each $k \geqslant 2$, the length function from $\mathrm{B}_k(\omega)$ to $\mathrm{B}_k(\omega)$ is in LOGSPACE.*

Proof: Given x on the input tape, to compute $|x|$ in k-ary, the machine first writes a 0 on the output tape, and then adds 1 in k-ary on the output tape each time it reads a symbol of x. We know that $|x|$ in k-ary is $O(\log |x|)$ by Lemma 2.1.2 (b).

□

In the proof of the next lemma, we give the modification, mentioned earlier, of the standard algorithm for k-ary multiplication, where $k \geqslant 2$.

Lemma 2.1.8. *Let $k \geqslant 2$. The multiplication function from $\mathrm{B}_k(\omega) \times \mathrm{B}_k(\omega)$ to $\mathrm{B}_k(\omega)$ is in LOGSPACE.*

Proof: Suppose we are given k-ary numbers x and y (written in reverse) on, respectively, the first and second input tapes of a Turing machine. Our proof that $x \cdot y$ can be computed using space no more than $O(\log(|x| + |y|))$ is in four steps. In the first three steps, we verify basic arithmetic properties of the length and the symbols of $x \cdot y$. We then go on to describe our machine's operation in the final step. We may assume that x and y are both nonzero; otherwise, the machine simply outputs 0.

Step 1. Since x and y are k-ary numbers, we must have $x \leqslant k^{|x|} - 1$ and $y \leqslant k^{|y|} - 1$. Hence $x \cdot y \leqslant (k^{|x|} - 1) \cdot (k^{|y|} - 1) = k^{|x|+|y|} - k^{|x|} - k^{|y|} + 1 \leqslant k^{|x|+|y|} - 1$, whose length is at most $|x| + |y|$. In other words, we have $|x \cdot y| \leqslant |x| + |y|$.

Step 2. For each $i \geqslant 0$, let x_i and y_i denote, respectively, the ith symbols of x and y. Here if $i \geqslant |x|$ (resp. $|y|$), then we let $x_i = 0$ (resp. $y_i = 0$), a convention that will prove useful in Step 3. We now have $x = \sum_{i=0}^{|x|-1} k^i x_i$ and $y = \sum_{i=0}^{|y|-1} k^i y_i$.

In order to arrive at an equation relating the symbols of $x \cdot y$ with the symbols of x and of y, consider the following recursive definitions:

- Let $m_0 = x_0y_0$, let $q_0 = \lfloor m_0/k \rfloor$, and let $r_0 = m_0 (\text{mod } k)$.
- For $i \geqslant 1$, let $m_i = (x_iy_0 + x_{i-1}y_1 + \cdots + x_1y_{i-1} + x_0y_i) + q_{i-1}$, let $q_i = \lfloor m_i/k \rfloor$, and let $r_i = m_i (\text{mod } k)$.

It is easy to see now that for each $i \geqslant 0$, the ith symbol of $x \cdot y$ is r_i.

Step 3. The algorithm we describe in Step 4 involves our machine explicitly writing down the m_i and the q_i on work tapes. Hence we must prove that the lengths of the m_i and the q_i are logarithmic in $|x| + |y|$. Our proof involves induction on i.

Since $x_i, y_i \leqslant k - 1$ for all $i \geqslant 0$, we have $m_0 \leqslant (k-1)^2$, and so $q_0 \leqslant \lfloor (k-1)^2/k \rfloor = \lfloor (k^2 - 2k + 1)/k \rfloor = \lfloor k - 2 + (1/k) \rfloor = k - 2$. We have $m_1 = q_0 + x_1y_0 + x_0y_1 \leqslant (k-2) + 2(k-1)^2 = 2k^2 - 3k = k(2k-3)$, and so $q_1 \leqslant 2k - 3$. Similarly we have $m_2 \leqslant k(3k-4)$ and $q_2 \leqslant 3k - 4$.

Suppose for $i \geqslant 1$ we have $m_i \leqslant k[(i+1)k - (i+2)]$ and hence $q_i \leqslant (i+1)k-(i+2)$. Then $m_{i+1} = q_i + \sum_{j=0}^{i+1} x_{i+1-j}y_j \leqslant (i+1)k-(i+2)+(i+2)(k-1)^2 = (i+1)k + (i+2)k^2 - (i+2)(2k) = (i+2)k^2 - (i+3)k = k[(i+2)k - (i+3)]$, and so $q_{i+1} \leqslant (i+2)k - (i+3)$. This completes the induction.

Now if $i \geqslant |x| + |y|$, then $m_i = 0 = q_i$ by our convention established in Step 2. So suppose $0 \leqslant i < |x| + |y|$. Then we have $m_i \leqslant k[(i+1)k - (i+2)] \leqslant k[(|x| + |y| + 1)k - (|x| + |y| + 2)]$. By Lemma 2.1.2 (a), it follows that $|m_i| = O(\log(|x| + |y|))$, and hence, we have $|q_i| = O(\log(|x| + |y|))$ also.

Step 4. In addition to the two input tapes and the output tape, our machine uses a counter tape and three other work tapes $W1$, $W2$, and $W3$. The machine begins by writing a 0 on the counter tape. Then it adds 1 in k-ary on the counter tape (using the standard algorithm) each time the first input cursor reads a symbol of x, and then each time the second input cursor reads a symbol of y. Step 1 assures us that at the end of this procedure, the counter tape has the maximum number of symbols possible for $x \cdot y$. Moreover, this procedure uses up space logarithmic in

$|x| + |y|$ by Lemma 2.1.2 (a). The input cursors now return to their initial extreme-left positions.

Now the machine is reading the first symbol x_0 of x and the first symbol y_0 of y. The machine's program contains explicit instructions as to what q_0 and r_0 (the first symbol of $x \cdot y$) are, and so it writes r_0 on the output tape and q_0 on tape $W1$. For example, if $k = 10$ and we are multiplying 1774 by 29, then $x_0 = 4$ and $y_0 = 9$. Based on its instructions, the machine writes 6 (which is r_0 in this case) on the output tape and 3 (i.e., q_0) on $W1$. Then the machine subtracts 1 in k-ary from the counter tape using the standard algorithm (to signify that we have already written the first symbol of $x \cdot y$). The first input cursor now moves right while the second input cursor moves left, which it cannot do, and therefore it simply points to y_0.

The machine is now reading x_1 and y_0, and based on its explicit instructions, writes $x_1 y_0$ on tape $W2$. In the example of multiplying 1774 by 29 above, we have $x_1 = 7$ and so the machine is instructed in this case to write 63 (in reverse) on $W2$. As soon as something is written on $W2$, the machine switches to a new state S, adds the contents of $W1$ and $W2$ in k-ary (which requires no space by Lemma 2.1.6), and writes the answer on $W3$. It then erases $W1$ and copies the contents of $W3$ on to $W1$. At this point, $W1$ contains $q_0 + x_1 y_0$ (which, in our example, is $3+63 = 66$). To obtain m_1 the machine needs to add $x_0 y_1$ to the number on $W1$. So it switches to a new state, and the first input cursor moves left while the second input cursor moves right. Now the machine is reading x_0 and y_1 and, once again, based on its explicit instructions, writes $x_0 y_1$ (which is 8 in our example since $y_1 = 2$) on $W2$ after erasing the previous contents of $W2$. Since W_2 has been written on, the machine switches to state S, adds the contents of $W1$ and $W2$, writing the answer (74 in our example) on $W3$ (after erasing $W3$), and then copies this answer on to $W1$ (after erasing $W1$). Then the machine switches state and the second input cursor moves right while the first input cursor moves left, which it cannot do.

As soon as the first input cursor cannot move left anymore, the machine "knows" that the number currently on $W3$ is some m_i. At this point in our example, $m_i = m_1$. Now r_1 (4 in our example) is simply the first symbol of m_1 (74 in our example), and q_1 is the rest of m_1. In fact, r_i is the first symbol of m_i and q_i is the rest of m_i for every m_i. So the machine switches to a new state, copies the first symbol on $W3$ on to the output tape, subtracts 1 on the counter tape, and copies the remaining symbols on $W3$ on to $W1$.

Now the machine must obtain m_2 and to do that, it must resume the "zigzag" motion of its input cursors. So the machine changes state, the second input cursor moves left while the first moves right. When the second cannot move left anymore, the machine starts writing and adding x_2y_0, x_1y_1, and x_0y_2 on the tapes $W1$, $W2$, and $W3$ in the manner described above for x_1y_0 and x_0y_1. Then it writes r_2 and q_2 when the first input cursor cannot move left anymore, again in the manner described for r_1 and q_1 . As soon as r_2 is written on the output tape, the machine subtracts 1 on the counter tape, and resumes the "zigzag" motion of its input cursors to obtain m_3.

The machine continues in this manner to obtain the remaining r_i, treating the ⊔s on either input tape as 0's, and stops when the content of the counter tape becomes zero. By Step 3, the lengths of the contents of $W1$, $W2$, and $W3$ will never be more than logarithmic in $|x| + |y|$.

□

In contrast to k-ary multiplication, it is not entirely clear to us how to carry out k-ary division for $k \geqslant 2$ in logarithmic space. So we cannot directly deal with the standard conversion of a number n written in a base b_1 to n written in another base b_2 that employs division by b_2. Instead we shall prove the result that $\mathrm{Bin}(\omega)$ is LOGSPACE set-isomorphic to the set $\mathrm{B}_k(\omega)$ for every $k \geqslant 3$ by producing isomorphisms between $\mathrm{Bin}(\omega)$ and certain other sets which will themselves be shown to be isomorphic to $\mathrm{B}_k(\omega)$, $k \geqslant 3$. Then these isomorphisms will be composed to produce the required isomorphism between $\mathrm{Bin}(\omega)$ and $\mathrm{B}_k(\omega)$, $k \geqslant 3$. In view of that, we examine the space complexity of compositions of functions in the next section.

2.2 Function Composition

We begin with a basic fact [23; Proposition 8.1] regarding the length of the output of a multi-variable function in a specified space complexity class.

Lemma 2.2.1. *Let $f(x_1, \ldots, x_n)$ be a function in $SPACE(f^*)$, where $n \geqslant 1$ and f^* is a proper complexity function. Let $|x| = |x_1| + \cdots + |x_n|$ denote the total length of an input n-tuple $(x_1, \ldots, x_n)$ to f. Then there are nonzero constants c and k such that f is in $TIME\left(c|x|^n 2^{kf^*(|x|)}\right)$, and hence $|f(x_1, \ldots, x_n)| \leqslant c|x|^n 2^{kf^*(|x|)}$.*

Proof: We may assume that there is a read-only and write-only Turing machine M that computes f, and suppose that M has T work tapes. Also suppose M operates using Q states and on S symbols. The maximum number of steps executed by M before halting is the total number of configurations possible for M. This is because if a configuration is repeated, M will enter a loop and never halt. A configuration of M is of the form $(q, u_1w_1, \ldots, u_nw_n, u_1^*w_1^*, \ldots, u_T^*w_T^*, v)$, where: (i) q is the current state of M, (ii) u_iw_i is the string on the ith input tape with the cursor pointing at the last symbol of u_i, (iii) $u_i^*w_i^*$ is the current string on the ith work tape with the cursor pointing at the last symbol of u_i^*, and (iv) v is the current string on the output tape. There are Q choices for q and less than $|x|$ choices for the cursor position on each of the n input tapes. Since strings on the work tapes cannot be longer than $f^*(|x|)$, there are $S^{f^*(|x|)}$ choices for each of the T work tape strings. Thus for the input and work tapes only, we have a total of at most $Q|x|^n S^{Tf^*(|x|)}$ possible configurations, each of which could result in M writing a symbol on the output tape.

□

Corollary 2.2.2. *Let $n \geqslant 1$, let f^* be a proper complexity function such that $f^*(x) > k\log(x)$ for every constant k and $x \in \omega$, and let $|x| = |x_1| + \cdots + |x_n|$. We have*

(a) *If $f(x_1, \ldots, x_n) \in$ CONSTANTSPACE, then $|f(x_1, \ldots, x_n)| \leqslant c|x|^n$ for some nonzero constant c.*

(b) *If $f(x_1, \ldots, x_n)$ is a LOGSPACE function, then $|f(x_1, \ldots, x_n)| \leqslant c|x|^r$ for some nonzero constants c and $r > n$.*

(c) *If $f(x_1, \ldots, x_n)$ is in SPACE(f^*), then $|f(x_1, \ldots, x_n)| \leqslant 2^{cf^*(|x|)}$ for some nonzero constant c.*

The next lemma (Proposition 8.2 in Papadimitriou [23]) shows that the space complexity class is preserved when two LOGSPACE functions are composed. We reproduce the argument, which, with appropriate changes, later becomes proofs in the cases of compositions where the complexity class is not preserved. We also use this argument in a modified form to prove a more general version of Lemma 2.2.3.

Lemma 2.2.3 (Space Composition Lemma I). *The composition $f \circ g$ of two LOGSPACE functions $g(x_1, \ldots, x_n)$ and $f(x)$, where $n \geqslant 1$, is in LOGSPACE.*

Proof: Suppose we are given $x_1, \ldots, x_n$ on the n input tapes. Let M_f and M_g be Turing machines that compute f and g, respectively, using up logarithmic space, and let $|x| = |x_1| + \cdots + |x_n|$.

We first note that in order to compute $(f \circ g)(x_1, \ldots, x_n)$, we cannot simply let M_g compute and write $g(x_1, \ldots, x_n)$ on some work tape(s) and then feed $g(x_1, \ldots, x_n)$ as input to M_f. This is because by Corollary 2.2.2 (b), the length of $g(x_1, \ldots, x_n)$ is not $O(\log|x|)$. In other words, the intermediate output $g(x_1, \ldots, x_n)$ may be too long to write out completely on a work tape. We overcome this difficulty by never writing all of $g(x_1, \ldots, x_n)$ on a tape. Instead, our machine M to compute $f \circ g$ simulates the operations of M_f and M_g in the special manner described in the next two paragraphs.

The program of M contains the programs of both M_f and M_g, and additional instructions. M begins by simulating M_f on certain work tapes. As soon as M_f "demands" to read the first symbol of $g(x_1, \dots, x_n)$, that is, right at the beginning, M changes state and simulates M_g on separate work tapes long enough to write down only the first symbol of $g(x_1, \dots, x_n)$ on the "input tape" for M_f. Once this is done, M changes state and writes a 1 in binary on a separate counter tape $T1$ to signify that the first symbol of $g(x_1, \dots, x_n)$ has been written so far. Then M simulates M_f again and "lets" M_f read the first symbol of $g(x_1, \dots, x_n)$. M continues to simulate M_f until M_f "demands" to read the second symbol of $g(x_1, \dots, x_n)$. At that point, M changes state, erases the first symbol of $g(x_1, \dots, x_n)$, and then simulates M_g long enough to output the second symbol of $g(x_1, \dots, x_n)$ on the place where the first symbol was written. After that, M adds 1 in binary on tape $T1$ and then simulates M_f to "let" M_f read the second symbol of $g(x_1, \dots, x_n)$. In general, when M masquerading as M_f "demands" to read the ith symbol of $g(x_1, \dots, x_n)$, M changes state and simulates M_g long enough to overwrite the $(i-1)$th symbol of $g(x_1, \dots, x_n)$ with the ith symbol, and then adds 1 on tape $T1$ so that $T1$ contains i in binary.

Since each symbol of $g(x_1, \dots, x_n)$ is overwritten with the next, a difficulty arises when M in the guise of M_f "demands" to read a previous symbol of $g(x_1, \dots, x_n)$ instead of the current ith symbol. In that situation, M subtracts 1 in binary on tape $T1$ and copies the resulting number $i - 1$ on to another counter tape $T2$. Then M starts simulating M_g *from the beginning* to overwrite the ith symbol of $g(x_1, \dots, x_n)$ with the *first* symbol of $g(x_1, \dots, x_n)$, then subtracts 1 on tape $T2$, then simulates M_g again long enough to overwrite the first symbol with the second symbol of $g(x_1, \dots, x_n)$, then subtracts 1 on $T2$, and so on, until there is a 0 on $T2$, at which point M, under the guise of M_f, can read the $(i-1)$th symbol of $g(x_1, \dots, x_n)$. This procedure can be repeated as often as necessary to allow M in the guise of M_f to read any previous symbol of $g(x_1, \dots, x_n)$ that has long been deleted.

It remains to show that M operates within space $O(\log|x|)$. Since each symbol of $g(x_1, \ldots, x_n)$ gets overwritten by the next one or the very first one, only constant space is used on that particular tape. On the tapes where M simulates M_g, only space $O(\log|x|)$ is used up since M_g operates in logarithmic space. Tapes $T1$ and $T2$ contain at most the total length of $g(x_1, \ldots, x_n)$ in *binary*. Thus the contents of these two tapes cannot be longer than $\log(c|x|^r)$ for some nonzero constants c and $r > n$, and $\log(c|x|^r) = O(\log|x|)$. Finally, the space used up by M while simulating M_f is logarithmic in the length of $g(x_1, \ldots, x_n)$, which, therefore, is at most $\log(c|x|^r) = O(\log|x|)$ as well.

$\square$

In the next six lemmas, the notation PSPACE $\circ$ LINSPACE $\subseteq$ EXSPACE, for example, means: The composition of a linear-space function of $n \geqslant 1$ variables followed by a polynomial-space function of one variable is an EXSPACE function. Except for the first one, these six lemmas are grouped according to the space complexity class of the function that is applied second in a composition.

Lemma 2.2.4 (Space Composition Lemma II). *Let f^* be a proper complexity function such that $f^*(x) \geqslant k \log x$ for every constant k and $x \in \omega$. We have*

(a) *CONSTANTSPACE $\circ$ CONSTANTSPACE $\subseteq$ LOGSPACE.*

(b) *CONSTANTSPACE $\circ$ SPACE(f^*) $\subseteq$ SPACE(f^*).*

(c) *SPACE(f^*) $\circ$ CONSTANTSPACE $\subseteq$ SPACE(f^*).*

Proof: (a) Given a CONSTANTSPACE function g of one variable and a CONSTANTSPACE function h of $n \geqslant 1$ variables with corresponding Turing machines M_g and M_h, our machine M computes $g \circ h$ by composing M_g and M_h in the manner described in the proof of the Space Composition Lemma I. While simulating M_g and M_h, M uses up constant space. Also M uses up the one "slot" on the tape needed to output h one symbol at a time. By Corollary 2.2.2, we have $|h(x)| \leqslant c|x|^n$, where c is some nonzero constant and $|x|$ is the total length of the input to h. Hence, on the binary counter tapes keeping track of the output symbols of h, M uses up space at most $\log|h(x)| \leqslant \log(c|x|^n)) = O(\log|x|)$.

(b) Given a function h of $n \geqslant 1$ variables in SPACE(f^*) and a single-variable CONSTANTSPACE function g with corresponding Turing machines M_h and M_g, our machine M once again computes $g \circ h$ by composing M_g and M_h in the manner described in the proof of the Space Composition Lemma I. This time M uses up space bounded by f^* while simulating M_h, the one "slot" on the tape needed to output h one symbol at a time, and constant space while simulating M_g. And by Corollary 2.2.2, this time we have $|h(x)| \leqslant 2^{cf^*(|x|)}$, where c and $|x|$ have the same meaning as in part (a). Hence, on the binary counter tapes M uses up space at most $\log\left(2^{cf^*(|x|)}\right) = O(f^*)$.

(c) The proof is similar to that of (b).

□

Lemma 2.2.5 (Space Composition Lemma III). *Let f^* be a proper complexity function such that $f^*(x) \geqslant k \log x$ for every constant k and $x \in \omega$. We have LOGSPACE $\circ$ SPACE(f^*) $\subseteq$ SPACE(f^*).*

Proof: Suppose we are given a LOGSPACE function g of one variable and a function h of $n \geqslant 1$ variables in SPACE(f^*) with corresponding Turing machines M_g and M_h. While computing $g \circ h$, the simulation of M_h uses up space bounded by f^* and also the one "slot" needed to output h one symbol at a time. Although the output of h is written one symbol at a time, the simulation of M_g utilizes the entire length of the output of h which, by Corollary 2.2.2, can be as long as $2^{cf^*(|x|)}$, where c is some nonzero constant and $|x|$ is the total length of the input to h. Since $g \in$ LOGSPACE, the simulation of M_g uses up space at most $\log\left(2^{cf^*(|x|)}\right)$, which is $O(f^*)$. Finally, the space used up in the binary counter tapes is at most $\log|h(x)| \leqslant \log\left(2^{cf^*(|x|)}\right)$ as well.

□

In the next lemma, we omit the two results PLOGSPACE $\circ$ LOGSPACE $\subseteq$ PLOGSPACE and PLOGSPACE $\circ$ CONSTANTSPACE $\subseteq$ PLOGSPACE, since they are covered by part (a) of the lemma.

Lemma 2.2.6 (Space Composition Lemma IV). *Let f^* be a proper complexity function such that $f^*(x) > kx$ for every constant k and $x \in \omega$. We have*

(a) *PLOGSPACE $\circ$ PLOGSPACE $\subseteq$ PLOGSPACE.*

(b) *PLOGSPACE $\circ$ LINSPACE $\subseteq$ PSPACE.*

(c) *PLOGSPACE $\circ$ SPACE(f^*) $\subseteq$ SPACE(f^*).*

Proof: (b) Suppose we are given a PLOGSPACE function g of one variable and a LINSPACE function h of $n \geqslant 1$ variables with corresponding Turing machines M_g and M_h. While computing $g \circ h$, the simulation of M_h uses up linear space and also the one "slot" needed to output h one symbol at a time. By Corollary 2.2.2, the output of h can be as long as $2^{c|x|}$, where c is some nonzero constant and $|x|$ is the total length of the input to h. Since $g \in$ PLOGSPACE, there is a nonzero constant r such that the simulation of M_g uses up space at most $\left(\log\left(2^{c|x|}\right)\right)^r$, which is polynomial in $|x|$. And the space used up in the binary counter tapes is at most $\log|h(x)| \leqslant \log\left(2^{c|x|}\right) = O(|x|)$.

The proofs of (a) and (c) are similar.

$\square$

We also omit results involving LINSPACE since they are entirely covered by the next lemma.

Lemma 2.2.7 (Space Composition Lemma V). *Let f^* be a proper complexity function such that $f^*(x) \geqslant k \log x$ for every constant k and $x \in \omega$. We have PSPACE $\circ$ SPACE (f^*) $\subseteq$ $\bigcup_k\{SPACE(2^{kf^*}) : k \geqslant 0\}$.*

Proof: Suppose we are given a PSPACE function g of one variable and a function h of $n \geqslant 1$ variables in SPACE (f^*) with corresponding Turing machines M_g and

M_h. While computing $g \circ h$, the simulation of M_h uses up space bounded by f^* and also the one "slot" needed to output h one symbol at a time. By Corollary 2.2.2, the output of h can be as long as $2^{cf^*(|x|)}$, where c is some nonzero constant and $|x|$ is the total length of the input to h. Since $g \in$ PSPACE, there is a nonzero constant r such that the simulation of M_g uses up space at most $\left(2^{cf^*(|x|)}\right)^r = 2^{rcf^*(|x|)}$. And the space used up in the binary counter tapes is at most $\log |h(x)| \leqslant \log \left(2^{cf^*(|x|)}\right) = O(f^*(|x|))$.

$\square$

As with the previous space composition lemmas, we shall omit results like SUPERPSPACE $\circ$ LOGSPACE $\subseteq$ SUPERPSPACE, which is covered by part (a) of the next lemma.

Lemma 2.2.8 (Space Composition Lemma VI). *Let f^* be a proper complexity function such that $f^*(x) \geqslant x^k$ for every constant k and $x \in \omega$. We have*

(a) *SUPERPSPACE $\circ$ PLOGSPACE $\subseteq$ SUPERPSPACE.*

(b) *SUPERPSPACE $\circ$ LINSPACE $\subseteq$ EXPSPACE.*

(c) *SUPERPSPACE $\circ$ SPACE(f^*) $\subseteq$ $\bigcup_k\{SPACE(2^{kf^*}) : k \geqslant 0\}$.*

Proof: (a) Suppose we are given a SUPERPSPACE function g of one variable and a PLOGSPACE function h of $n \geqslant 1$ variables with corresponding Turing machines M_g and M_h. While computing $g \circ h$, the simulation of M_h uses up poly-logarithmic space and also the one "slot" needed to output h one symbol at a time. By Corollary 2.2.2, the output of h can be as long as $2^{c(\log|x|)^d}$, where c and d are some nonzero constants and $|x|$ is the total length of the input to h. Since $g \in$ SUPERPSPACE, there is a nonzero constant r such that the simulation of M_g uses up space at most $2^{\left(\log\left(2^{c(\log|x|)^d}\right)\right)^r}$, which is super-polynomial in $|x|$. And the space used up in the binary counter tapes is at most $\log |h(x)| \leqslant \log \left(2^{c(\log|x|)^d}\right)$, which is poly-logarithmic in $|x|$.

The proofs of (b) and (c) are similar.

$\square$

We omit any lemmas about EXSPACE and deal with EXPSPACE instead in the next lemma. This is because *Lemma 2.2.9 still holds if we replace EXPSPACE with EXSPACE in each of the statements* (a)–(d).

Lemma 2.2.9. (Space Composition Lemma VII)

(a) *EXPSPACE* $\circ$ *LOGSPACE* $\subseteq$ *EXPSPACE.*

(b) *EXPSPACE* $\circ$ *PLOGSPACE* $\subseteq$ *EXSUPERPSPACE.*

(c) *EXPSPACE* $\circ$ *LINSPACE* $\subseteq$ *DOUBEXSPACE.*

(d) *EXPSPACE* $\circ$ *PSPACE* $\subseteq$ *DOUBEXPSPACE.*

Proof: (a) Given an EXPSPACE function g of one variable and a LOGSPACE function h of $n \geqslant 1$ variables with corresponding Turing machines M_g and M_h, the length of the output of h is at most $c|x|^r$ for some nonzero constants c and $r > n$. And so there is a nonzero constant d such that the simulation of M_g uses up space at most $2^{(c|x|^r)^d}$, which is exponentially polynomial in $|x|$. The simulation of M_h and utilization of the binary counter uses up even less space.

The proofs of (b)–(d) are similar.

□

We now proceed to generalize the Space Composition Lemma I to include functions whose outputs are many-dimensional vectors rather than simply scalars. In order to facilitate this generalization, we identify the vector $\vec{x} = \langle x_1, \ldots, x_n \rangle$ with the value $\rho(x_1, \ldots, x_n)$ of the coding function ρ on the n-tuple $(x_1, \ldots, x_n)$, as defined in the following lemma.

Lemma 2.2.10 (Coding Lemma). *Let* $x_1, \ldots, x_n$ *be finite strings of letters of some finite alphabet* Σ*, and suppose* $x_1 = e^1_1 e^1_2 \ldots e^1_{k_1}$, $x_2 = e^2_1 e^2_2 \ldots e^2_{k_2}$, $\ldots$, $x_n = e^n_1 e^n_2 \ldots e^n_{k_n}$. *Define* $\rho : (\Sigma^*)^n \to (\Sigma \cup \{0, 1\})^*$ *as follows :*

$$\rho(x_1, \ldots, x_n) = e^1_1 1 e^1_2 1 \ldots 1 e^1_{k_1} 0 e^2_1 1 e^2_2 1 \ldots 1 e^2_{k_2} 0 \ldots 0 e^n_1 1 e^n_2 1 \ldots 1 e^n_{k_n}.$$

Then both ρ *and* ρ^{-1} *are in ZEROSPACE.*

Proof: Given $x_1, \ldots, x_n$ on n input tapes, our machine begins by copying each symbol e_i^1 of x_1 followed by a 1 on to the output tape. When the machine encounters the $\sqcup$ on the first input tape, it writes a 0 on the output tape. It then repeats the procedure while reading x_2 on the second input tape, and continues this procedure for $x_3, \ldots, x_n$.

The machine for ρ^{-1} has n output tapes. Suppose we are given the string $e_1^1 1 e_2^1 1 \ldots 1 e_{k_1}^1 0 e_1^2 1 e_2^2 1 \ldots 1 e_{k_2}^2 0 \ldots 0 e_1^n 1 e_2^n 1 \ldots 1 e_{k_n}^n$ on the machine's input tape. The machine begins by copying the first input symbol on to the first output tape, then reads the second input symbol, then copies the third input symbol on to the first output tape, and so on, until it encounters a 0 after having just copied an input symbol. At this point, the machine switches to a new state and repeats the procedure on the second output tape while reading the input starting at e_1^2. In this way, the machine ends up using n different states to write the n strings $x_1, \ldots, x_n$ on the n output tapes.

$\square$

In the next lemma, we show that if $\vec{x} = \langle x_1, \ldots, x_n \rangle$, and h and H are functions such that $h(x_1, \ldots, x_n) = H(\vec{x})$, then h and H almost always belong to the same space complexity class.

Lemma 2.2.11. *Let $x_1, \ldots, x_n$ be strings in some Σ^* and identify the vector $\vec{x} = \langle x_1, \ldots, x_n \rangle$ with its code $\rho(x_1, \ldots, x_n)$. Suppose h and H are functions such that $h(x_1, \ldots, x_n) = H(\vec{x})$. Let g^* be a* nonconstant *proper complexity function. Then we have*

(a) *If $H \in$ CONSTANTSPACE, then $h \in$ LOGSPACE.*

(b) *If $H \in$ SPACE(g^*), then $h \in$ SPACE(g^*).*

(c) *If $h \in$ CONSTANTSPACE, then $H \in$ LOGSPACE.*

(d) *If $h \in$ SPACE(g^*), then $H \in$ SPACE(g^*).*

Proof: (a) To compute $h(x_1, \ldots, x_n)$, we compose the ZEROSPACE conversion of $(x_1, \ldots, x_n)$ to $\rho(x_1, \ldots, x_n)$, that is, $\vec{x}$, followed by the CONSTANTSPACE computation of $H(\vec{x})$. We know that this composition is in LOGSPACE by the Space Composition Lemma II (a).

(b) We argue exactly as in part (a) but invoke part (c) of the Space Composition Lemma II.

(c) The argument here is that in part (d), *mutatis mutandis.* Hence, we give the argument for part (d) in full detail.

(d) Let M be a machine with one input tape and suppose we are given $\rho(x_1, \ldots, x_n)$ on this input tape. We describe how M can compute the output $H(\langle x_1, \ldots, x_n \rangle) = H(\rho(x_1, \ldots, x_n))$ while operating in SPACE(g^*).

First, let $M_{\rho^{-1}}$ be the machine in the proof of the Coding Lemma that outputs $x_1, \ldots, x_n$ on its n output tapes (without using up any space) when given $\rho(x_1, \ldots, x_n)$ on its input tape. For convenience, we assume further that $M_{\rho^{-1}}$ has been modified so that instead of outputting the *complete* strings $x_1, \ldots, x_n$ on its n output tapes, $M_{\rho^{-1}}$ actually *overwrites* each symbol e^i_j of each x_i on its ith output tape by the next symbol e^i_{j+1} of x_i. With this modification, $M_{\rho^{-1}}$ still operates in ZEROSPACE but each of its output tapes contain at most one symbol at any one time. Finally, let M_h be a machine that outputs $h(x_1, \ldots, x_n)$ when given $x_1, \ldots, x_n$ on its n input tapes, and operates in SPACE (g^*).

The machine M has $3n$ work tapes T^1_1, T^1_2, W_1, T^2_1, T^2_2, W_2, $\ldots$, T^n_1, T^n_2, W_n, among others. The n work tapes $W_1, \ldots W_n$ serve as output tapes for $M_{\rho^{-1}}$, (i.e., input tapes for M_h). As for the remaining labeled work tapes T^1_1, T^1_2, T^2_1, T^2_2, $\ldots$, T^n_1, T^n_2, each of the tapes T^i_1 and T^i_2 serve as binary counter tapes for the intermediate output tape W_i. The machine M computes the output $H(\langle x_1, \ldots, x_n \rangle) = H(\rho(x_1, \ldots, x_n))$ by composing $M_{\rho^{-1}}$ followed by M_h in a manner similar to that of the machine described in the proof of the Space Composition Lemma I. *We need to* *be careful about the requirement that M_h can read n symbols, one on each of its n input tapes, all at the same time.*

In the beginning, M writes 0s on each of the $2n$ counter tapes T_j^i. Then M simulates M_h. At this point, each input cursor of M_h either moves one place to the right or remains statinary. For each $i \in \{1, \ldots, n\}$, if the ith input cursor of M_h "wants" to move to the right, then M adds 1 in binary on the counter tape T_1^i. But if the ith input cursor of M_h "wants" to remain stationary, then M does nothing on tape T_1^i. Then M copies the contents of each tape T_1^i on to tape T_2^i. Now if there is a 1 on tape T_2^1, then M simulates $M_{\rho^{-1}}$ long enough to write the first symbol of x_1 on tape W_1. After that, M simulates $M_{\rho^{-1}}$ in a modified form long enough to output all the symbols of x_1 without actually overwriting the first symbol of x_1. However, if there is a 0 on tape T_2^1, then M simulates $M_{\rho^{-1}}$ in a modified form long enough to exhaust all the symbols of x_1 without actually writing anything on W_1. Similarly, the first symbol of each of the remaining x_i is either written or not written on tape W_i depending on whether there is a 0 or a 1 on tape T_2^i. Now M can simulate M_h again to read the symbols on the tapes W_i all at the same time and do whatever M_h "wants" to do.

Now each input cursor of M_h may "want" to move one place to the right or to the left or remain stationary. If an input cursor wants to move right (resp. left), M adds (resp. subtracts) 1 in binary on the corresponding counter tape T_1^i. But if this input cursor "wants" to remain stationary, then nothing is done on the corresponding counter tape. Then M copies the contents of each tape T_1^i on to tape T_2^i. Now M simulates $M_{\rho^{-1}}$ from the beginning, and each time a symbol of x_1 gets written on W_1, a 1 is subtracted in binary on tape T_2^1 until the content of T_2^1 becomes 0. Now W_1 contains the symbol σ of x_1 that M_h "wanted" to read. So now M simulates $M_{\rho^{-1}}$ in a modified form long enough to output all the remaining symbols of x_1 without actually overwriting σ. Then M continues to simulate $M_{\rho^{-1}}$ and each time a symbol of x_2 gets written on W_2, a 1 is subtracted in binary on tape T_2^2 until the content of T_2^2 becomes 0. This process continues for the remaining x_i, at the end of which M can simulate M_h again to read the symbols on the tapes W_i all at the same time and do whatever M_h "wants" to do. Finally, M halts when M_h "wants" to halt.

It remains to show that M operates in SPACE(g^*). Evidently the length of the contents of each of the binary counter tapes T_j^i is logarithmic in the length of the input to M. The length of the contents of each of the tapes W_i is at most 1. And M_h operates in SPACE(g^*). Since g^* is a nonconstant proper complexity function, we have $g^*(n) \geqslant k \log n$ for every constant k and $n \in \omega$. It now follows that M operates in SPACE(g^*).

$\square$

We have now finally reached the position where we can generalize the Space Composition Lemma I.

Lemma 2.2.12 (Generalized Space Composition Lemma). *Let $x_1, \ldots, x_n$ be strings in some Σ^*. Let $h_1, \ldots, h_m$ be functions of n variables, and let g be a function of m variables. Suppose that each h_i is in SPACE(H^*) and that g is in SPACE(G^*), where G^* and H^* are* nonconstant *proper complexity functions. Let f be the composition defined by $f(x_1, \ldots, x_n) = g(h_1(x_1, \ldots, x_n), \ldots, h_m(x_1, \ldots, x_n))$. Then f is in SPACE($G^*(2^{cH^*})$) for some nonzero constant c.*

Proof: Let $y_1, \ldots, y_m$ be arbitrary strings in Σ^*. We recall our usual identifications $\vec{x} = \langle x_1, \ldots, x_n \rangle = \rho(x_1, \ldots, x_n)$ and $\vec{y} = \langle y_1, \ldots, y_m \rangle = \rho(y_1, \ldots, y_m)$. For each $i \in \{1, \ldots, m\}$, let $H_i(\vec{x}) = h_i(x_1, \ldots, x_n)$, and let $G(\vec{y}) = g(y_1, \ldots, y_m)$. By the previous lemma, each H_i is in SPACE(H^*) and G is in SPACE(G^*). Note that $f(x_1, \ldots, x_n) = G(\langle H_1(\vec{x}), \ldots, H_m(\vec{x}) \rangle)$. Since ρ is in ZEROSPACE and we have $\langle H_1(\vec{x}), \ldots, H_m(\vec{x}) \rangle = \rho(H_1(\vec{x}), \ldots, H_m(\vec{x}))$, the Space Composition Lemma II and an argument similar to that in the proof of Lemma 2.2.11 (d) shows that $\langle H_1(\vec{x}), \ldots, H_m(\vec{x}) \rangle$ is in SPACE(H^*). Therefore by Lemma 2.2.1, we have the inequality $|\langle H_1(\vec{x}), \ldots, H_m(\vec{x})\rangle| \leqslant 2^{cH^*(|x|)}$, where c is some nonzero constant and $|x| = |x_1| + \cdots + |x_n|$. Since G^* is a *nonconstant* proper complexity function, an argument similar to that in the proof of the Space Composition Lemma I now shows that $f = G \circ H$ is in SPACE$(G^*(2^{cH^*}))$.

□

2.3 Applications of Function Composition

As an immediate application of the Space Composition Lemmas, we have the following:

Lemma 2.3.1. *Let ω be the set of natural numbers in either the tally or the binary representation. The exponential function $f(x) = 2^x$ from ω to ω is in LINSPACE, while the doubled exponential function $g(x) = 2^{2^x}$ from ω to ω is in EXSPACE.*

Proof: Note that $g = f \circ f$, and so if $f \in$ LINSPACE, then $g \in$ EXSPACE by the Space Composition Lemma V.

To prove that $f \in$ LINSPACE, first let $h : \mathrm{Tal}(\omega) \to \mathrm{Bin}(\omega)$ be defined by $h(x) = 0^{|x|}{}^\frown 1$. Thus h computes bin(2^x) given input $x \in \mathrm{Tal}(\omega)$. Moreover, we have $h \in$ ZEROSPACE since given $x \in \mathrm{Tal}(\omega)$ on an input tape, a Turing machine can output $h(x)$ by writing 0's on its output tape each time it reads a 1 of x and then writing a 1 on the output tape when it encounters the $\sqcup$ on the input tape. Finally, let μ_2 be as in Lemma 2.1.2 (c), that is, μ_2 converts a binary number to tally. Then $\mu_2 \in$ LINSPACE.

Now the exponential function $f : \mathrm{Tal}(\omega) \to \mathrm{Tal}(\omega)$ is $\mu_2 \circ h$, which is in LINSPACE by the Space Composition Lemma II. And the exponential function $f : \mathrm{Bin}(\omega) \to \mathrm{Bin}(\omega)$ is $h \circ \mu_2$, which is also in LINSPACE by the Space Composition Lemma II.

□

Next, we give two applications dealing with bijections from $\omega \times \omega$ to ω. The first bijection we consider happens to be the standard one. Note that we are unable to replace $\mathrm{Tal}(\omega)$ with $\mathrm{Bin}(\omega)$ in part (b) of Lemma 2.3.2.

Lemma 2.3.2. *Let ω be the set of natural numbers in either the tally or the binary representation.*

(a) *The pairing function $[\cdot,\cdot]$ from $\omega \times \omega$ to ω defined by $[x,y] = \frac{1}{2}[(x+y)^2 + 3x + y]$ is in LOGSPACE.*

(b) *The inverse $f : \mathrm{Tal}(\omega) \to \omega \times \omega$ of the pairing function is in LOGSPACE. This inverse is given explicitly by the formula $f(m) = (\phi(m), \psi(m))$ if $n(n+1)/2 \leqslant m < (n+1)(n+2)/2$, where $\phi(m) = m - [n(n+1)/2]$ and $\psi(m) = [(n+1)(n+2)/2] - m - 1$.*

Proof: (a) Firs,t let ω have the tally representation. Define the function $g : \mathrm{Tal}(\omega) \times \mathrm{Tal}(\omega) \to \mathrm{Tal}(\omega)$ by $g(x,y) = \frac{1}{2}(x+y)$. Then g is in LOGSPACE, being the composition of tally addition and division, either of which is in ZEROSPACE. Now define the functions h_1 and h_2 from $\mathrm{Tal}(\omega) \times \mathrm{Tal}(\omega)$ to $\mathrm{Tal}(\omega)$ by $h_1(x,y) = (x+y)^2$ and $h_2(x,y) = 3x + y$. Then h_2 is in ZEROSPACE since given x and y in tally on two input tapes, our machine uses three states to copy x on to the output tape three times, and then copies y on to the output tape. And h_1 is in LOGSPACE because given x and y in tally on two input tapes, our machine adds 1 in binary on a work tape T each time it reads a 1 of x and then of y, resulting in $\mathrm{bin}(x+y)$ on T. Then each time it subtracts a 1 in binary on T, the machine copies x and then y on to the output tape. We now have $[x,y] = g(h_1(x,y), h_2(x,y))$, and since g, h_1, and h_2 are all in LOGSPACE, it follows from the Generalized Space Composition Lemma that the pairing function is in LOGSPACE.

Now let ω have the binary representation. Define g, h_1, and h_2 as above but this time define g only on the set of pairs of numbers whose sum is even. Then g is in ZEROSPACE. This is because given binary numbers x and y on two input tapes such that $x + y$ is even, our machine simulates the addition of x to y, as described in the proof of Lemma 2.1.6, but this time it simply does not write the first symbol (which will be a 0) of $x + y$ on the output tape. To see that h_1 and h_2 are in LOGSPACE, we first define f_1, f_2, f_3, and f_4 from $\mathrm{Bin}(\omega) \times \mathrm{Bin}(\omega)$ to $\mathrm{Bin}(\omega)$ by $f_1(x, y) = x + y$, $f_2(x, y) = xy$, $f_3(x, y) = 3x$, and $f_4(x, y) = y$. Evidently f_4 is in ZEROSPACE. We have $f_1 \in$ ZEROSPACE by Lemma 2.1.6, while f_2 and f_3 are in LOGSPACE by Lemma 2.1.8. (The machine for $f_3(x, y)$ simply simulates the one for $f_2(x, y)$ on inputs x and 3.) We now have $h_1(x, y) = f_2(f_1(x, y), f_1(x, y))$ and $h_2(x, y) = f_1(f_3(x, y), f_4(x, y))$. Since the f_i are all in LOGSPACE, it follows from the Generalized Space Composition Lemma that h_1 and h_2 are in LOGSPACE. The same lemma now shows that $[x, y] = g(h_1(x, y), h_2(x, y))$ is in LOGSPACE.

(b) Given $m \in \omega$ in tally on the input tape, our machine has to output $\phi(m) \sqcup \psi(m)$ on its output tape. In order to do that, it must determine the n such that $n(n+1)/2 \leqslant m < (n+1)(n+2)/2$. In fact, our machine computes the number $n(n+1)/2$ which is $\leqslant m$ using four work tapes $T1$–$T4$ and writes $n(n+1)/2$ on $T4$. *All the work done on tapes $T1$–$T4$ will be in binary.* If $m = 0$ (resp. 1), then $n = 0$ (resp. 1), and so $n(n+1)/2 = 0$ (resp. 1). Hence the machine immediately writes a 0 (resp. 1) on $T4$. Otherwise, the machine writes a 0 on $T1$ and a 1 on $T2$, and then performs the following procedure: It simulates the addition of the contents of $T1$ and $T2$, and writes the answer on $T3$. After that, the machine copies the contents of $T3$ on to $T1$ and also on to $T4$. Once this is completed, the machine advances the input cursor (starting in its extreme-left position) one place to the write each time a 1 is subtracted on $T4$. If the input cursor reads a 1 of m when the content of $T4$ becomes 0, then the correct number $0 + 1 + \cdots + n = n(n+1)/2 \leqslant m$ may still not have been written on $T4$, and so the input cursor goes back to its extreme-left position, the machine adds 1 on tape $T2$, and the above procedure is repeated. But if the input cursor reads a $\sqcup$ when the content of $T4$ becomes 0, then $T4$ contained the correct $n(n+1)/2$, that is, $T1$ now contains the correct $n(n+1)/2$, and so the machine copies the contents of $T1$ on to $T4$. On a separate worktape $T5$, the machine writes m in binary. After that, the machine simulates the binary subtraction of the contents of $T4$ from $T5$, writing the answer on another work tape $T6$. Note that $T6$ now contains $\text{bin}(\phi(m))$. The machine now copies this $\text{bin}(\phi(m))$ on to the output tape if the output is to be in $\text{Bin}(\omega) \times \text{Bin}(\omega)$. Otherwise, the machine writes a 1 on the output tape each time it subtracts a 1 in binary on $T6$. At the end of either of these maneuvers, the machine writes a $\sqcup$ on the output tape.

It remains to output $\psi(m)$. Recall that at this point, $T1$ and $T4$ contain the correct number $0 + 1 + \cdots + n = n(n+1)/2 \leqslant m$. To obtain $(n+1)(n+2)/2$, the machine adds 1 on $T2$, and then adds the contents of $T1$ and $T2$, writing the answer on $T3$. Now $T3$ has $0 + 1 + \cdots + n + (n+1) = (n+1)(n+2)/2$. Meanwhile, $T5$ contains $\text{bin}(m)$. The machine adds 1 on $T5$ and then simulates the binary subtraction of the contents of $T5$ from $T3$, writing the answer, that is, $\text{bin}(\psi(m))$, on $T6$. The machine now copies this $\text{bin}(\psi(m))$ on to the output tape if the output is to be in $\text{Bin}(\omega) \times \text{Bin}(\omega)$. Otherwise, the machine writes a 1 on the output tape each time it subtracts a 1 in binary on $T6$.

$\square$

Although we cannot prove that the standard bijection from $\omega \times \omega$ to ω in the previous lemma is in LOGSPACE, we are able to construct another such bijection in the next lemma that does happen to be in LOGSPACE. Using this new bijection, we avoid the problem of having to explicitly right down the number $0 + 1 + \cdots + n = n(n+1)/2 \leqslant m$ in binary when the input m itself is in binary, which is the reason why we cannot prove that the standard bijection from $\omega \times \omega$ to ω is in LOGSPACE.

Lemma 2.3.3. *Let ω be the set of natural numbers in either the tally or the binary representation. For each $n \geqslant 1$, define*

$$\begin{aligned} A_n &= \{0, 1, \ldots, 2^n - 1\} \times \{2^n, 2^n + 1, \ldots, 2^{n+1} - 1\}, \\ B_n &= \{2^n, 2^n + 1, \ldots, 2^{n+1} - 1\} \times \{0, 1, \ldots, 2^n - 1\}, \\ C_n &= \{2^n, 2^n + 1, \ldots, 2^{n+1} - 1\} \times \{2^n, 2^n + 1, \ldots, 2^{n+1} - 1\}. \end{aligned}$$

(a) *Let $f : \omega \times \omega \to \omega$ be defined by the following set of rules:* $(0,0) \mapsto 0$, $(0,1) \mapsto 1$, $(1,0) \mapsto 2$, $(1,1) \mapsto 3$. *And for each $n \geqslant 1$, we define*

$$\begin{array}{ll} (x,y) \mapsto 2^n x + y + 2^{2n} - 2^n & \text{if } (x,y) \in A_n, \\ (x,y) \mapsto 2^n x + y + 2^{2n} & \text{if } (x,y) \in B_n, \\ (x,y) \mapsto 2^n x + y + 2^{2n+1} - 2^n & \text{if } (x,y) \in C_n. \end{array}$$

Then f is both one-to-one and onto, and in LOGSPACE.

(b) *The inverse $g : \omega \to \omega \times \omega$ of f is also in LOGSPACE. This inverse is given explicitly as follows:* $0 \mapsto (0,0)$, $1 \mapsto (0,1)$, $2 \mapsto (1,0)$, $3 \mapsto (1,1)$. *And for each $z \in \omega \setminus \{0,1,2,3\}$ and the corresponding $n \geqslant 1$, we have $g(z) = (x,y)$, where*

$$\begin{array}{lll} x = \left\lfloor \dfrac{z - 2^{2n}}{2^n} \right\rfloor, & & \\ & \text{if} & 2^{2n} \leqslant z \leqslant 2 \cdot 2^{2n} - 1, \\ y = (z - 2^{2n}) (\text{mod } 2^n) + 2^n & & \end{array}$$

$$\begin{array}{lll} x = \left\lfloor \dfrac{z - 2^{2n}}{2^n} \right\rfloor, & & \\ & \text{if} & 2 \cdot 2^{2n} \leqslant z \leqslant 3 \cdot 2^{2n} - 1, \\ y = (z - 2^{2n}) (\text{mod } 2^n) & & \end{array}$$

$$\begin{array}{lll} x = \left\lfloor \dfrac{z - 2^{2n}}{2^n} \right\rfloor - 2^n, & & \\ & \text{if} & 3 \cdot 2^{2n} \leqslant z \leqslant 4 \cdot 2^{2n} - 1. \\ y = (z - 2^{2n}) (\text{mod } 2^n) + 2^n & & \end{array}$$

Proof: (a) First, we show that f is indeed defined on all of $\omega \times \omega$ and that f is a bijection onto ω.

For each $n \geqslant 1$, the smallest element of the set $f(A_n)$ is $2^n x + y + 2^{2n} - 2^n = 2^n(0) + (2^n) + 2^{2n} - 2^n = 2^{2n}$ (which is 4 if n=1), while its largest element is $2^n x + y + 2^{2n} - 2^n = 2^n(2^n - 1) + (2^{n+1} - 1) + 2^{2n} - 2^n = 2 \cdot 2^{2n} - 1$. Similarly, the smallest element of the set $f(B_n)$ is $2 \cdot 2^{2n}$, while its largest element is $3 \cdot 2^{2n} - 1$. And the smallest element of $f(C_n)$ is $3 \cdot 2^{2n}$, while its largest element is $4 \cdot 2^{2n} - 1$. Thus for each $n \geqslant 1$, the sets $f(A_n)$, $f(B_n)$, and $f(C_n)$ are mutually disjoint.

Now $f\restriction_{A_n}$ for $n \geqslant 1$ is one-to-one because $2^n(x_1) + (y_1) + 2^{2n} - 2^n = 2^n(x_2) + (y_2) + 2^{2n} - 2^n \implies 2^n(x_1 - x_2) = y_2 - y_1 \leqslant 2^{n+1} - 1 - 2^n \implies (x_1 - x_2) \leqslant 1 - (1/2^n) < 1 \implies x_1 = x_2 \implies y_1 = y_2$. Similarly, $f\restriction_{B_n}$ and $f\restriction_{C_n}$ are also one-to-one. Thus for each $n \geqslant 1$, we have $|A_n| = |f(A_n)|$, $|B_n| = |f(B_n)|$, and $|C_n| = |f(C_n)|$. It now follows that f is defined on all of $\omega \times \omega$ and that f is onto ω.

We now show that f is in LOGSPACE.

Suppose we are given x and y on the two input tapes of our machine. If $(x, y) \in \{0,1\} \times \{0,1\}$, then the output is immediate. So assume that $(x, y) \notin \{0,1\} \times \{0,1\}$. If either x and y are in tally, then our machine first writes down $\text{bin}(x)$ and $\text{bin}(y)$ on two separate worktapes and treats these binary numbers as the inputs from now on. The machine computes the $n \geqslant 1$ such that $(x, y) \in A_n$ or B_n or C_n. We recall that $2^{|\text{bin}(x)|-1} \leqslant x \leqslant 2^{|\text{bin}(x)|} - 1$ and $2^{|\text{bin}(y)|-1} \leqslant y \leqslant 2^{|\text{bin}(y)|} - 1$. Hence, our machine only has three straightforward cases to consider: (i) If $|\text{bin}(x)| = |\text{bin}(y)|$, then $n = |\text{bin}(x)| - 1 = |\text{bin}(y)| - 1$, and $(x, y) \in C_n$. (ii) If $|\text{bin}(x)| < |\text{bin}(y)|$, then $n = |\text{bin}(y)| - 1$, and we claim that $(x, y) \in A_n$. Otherwise, $x > 2^n - 1$, i.e., $x \geqslant 2^n = 2^{|\text{bin}(y)|-1}$, which implies $|\text{bin}(x)| \geqslant |\text{bin}(y)|$, a contradiction. (iii) Analogously to the second case, if $|\text{bin}(x)| > |\text{bin}(y)|$, then $n = |\text{bin}(x)| - 1$ and $(x, y) \in B_n$. Our machine now writes $\text{bin}(n)$ on a separate worktape. We note that the procedures described so far all use up at most logarithmic space.

Now that our machine "knows" whether $(x, y) \in A_n$ or B_n or C_n, it proceeds to compute $f(x, y)$ in binary using the appropriate formula. We shall illustrate how the machine can do this using up at most logarithmic space in the case where $(x, y) \in A_n$. The other two cases are similar.

If $(x, y) \in A_n$, the machine must output the binary number $f(x, y) = 2^{\text{bin}(n)} \cdot \text{bin}(x) + \text{bin}(y) + 2^{2 \cdot \text{bin}(n)} - 2^{\text{bin}(n)}$. We will define LOGSPACE functions f_1, f_2, f_3, and f_4 such that $f(x, y) = f_4(f_1(x, y), f_2(x, y), f_3(x, y))$. It will then follow from the Generalized Space Composition Lemma that $f(x, y)$ is in LOGSPACE. And to output $f(x, y)$ in tally, the machine simply converts $\text{bin}(f(x, y))$ to tally. This conversion is linear in $\text{bin}(x)$ and $\text{bin}(y)$, and hence logarithmic if the inputs x and y are originally given in tally.

First let $f_4(\text{bin}(a), \text{bin}(b), \text{bin}(c)) = \text{bin}(a) + \text{bin}(b) + \text{bin}(c)$. Then $f_4 \in$ LOGSPACE by the Space Composition Lemma I. Now define $f_1(\text{bin}(x), \text{bin}(y)) = 2^{\text{bin}(n)} \cdot \text{bin}(x)$. Then f_1 is in LOGSPACE because the output is n zeros followed by $\text{bin}(x)$, and so the machine simply copies $\text{bin}(n)$ on a separate worktape, writes a zero on the "output tape" of f_1 each time it subtracts a 1 in binary from $\text{bin}(n)$, and then copies $\text{bin}(x)$ to the right of the zeros. The function $f_2(\text{bin}(x), \text{bin}(y)) = \text{bin}(y)$ is in ZEROSPACE. And finally, the function $f_3(\text{bin}(x), \text{bin}(y)) = 2^{2\text{bin}(n)} - 2^{\text{bin}(n)}$ is in LOGSPACE since $f_3(\text{bin}(x), \text{bin}(y)) = f_7(f_5(\text{bin}(x), \text{bin}(y)), f_6(\text{bin}(x), \text{bin}(y)))$, where we have f_5, f_6, and f_7 as follows: $f_7(\text{bin}(a), \text{bin}(b)) = \text{bin}(a) - \text{bin}(b) \in$ LOGSPACE, while $f_5(\text{bin}(x), \text{bin}(y)) = 2^{2\cdot\text{bin}(n)} \in$ LOGSPACE because the output is $2n$ zeros followed by a 1, and $2n$ zeros can be written given $\text{bin}(n)$ using up only logarithmic space, and $f_6(\text{bin}(x), \text{bin}(y)) = 2^{\text{bin}(n)} \in$ LOGSPACE, again because the output is n zeros followed by a 1. We now have $f(x, y) = f_4(f_1(x, y), f_2(x, y), f_3(x, y))$ in binary, and this function is in LOGSPACE by the Generalized Space Composition Lemma.

(b) First we must verify that $f \circ g = id$. If $z \in \{0, 1, 2, 3\}$, then certainly $(f \circ g)(z) = z$. Now let $z \in \omega \setminus \{0, 1, 2, 3\}$ and suppose there exists an $n \geqslant 1$ such that $2^{2n} \leqslant z \leqslant 2 \cdot 2^{2n} - 1$. Then $g(z) = (x, y)$, where $x = \lfloor (z - 2^{2n})/2^n \rfloor$ and $y = (z - 2^{2n})(\text{mod } 2^n) + 2^n$. Hence $f(g(z)) = 2^n x + y + 2^{2n} - 2^n = \{2^n \lfloor (z - 2^{2n})/2^n \rfloor + (z - 2^{2n})(\text{mod } 2^n)\} + 2^n + 2^{2n} - 2^n = \{z - 2^{2n}\} + 2^{2n} = z$. The argument for the cases where $2 \cdot 2^{2n} \leqslant z \leqslant 3 \cdot 2^{2n} - 1$ and $3 \cdot 2^{2n} \leqslant z \leqslant 4 \cdot 2^{2n} - 1$ are similar.

Now we prove that $g \in$ LOGSPACE. If the input z to our machine is in $\{0, 1, 2, 3\}$, then the output is immediate. Otherwise, if z is in tally, then the machine first writes $\text{bin}(z)$ on a separate work tape and treats $\text{bin}(z)$ as the input from now on. Our machine then proceeds to compute the $n \geqslant 1$ such that $2^{2n} \leqslant z \leqslant 4 \cdot 2^{2n} - 1 = 2^{2(n+1)} - 1$. Since $2^{|\text{bin}(z)|-1} \leqslant z \leqslant 2^{|\text{bin}(z)|} - 1$, we have $|\text{bin}(z)| = 2n + 1$ or $|\text{bin}(z)| = 2(n + 1)$. Hence $n = (|\text{bin}(z)| - 1)/2$ if $|\text{bin}(z)|$ is odd, and $n = (|\mathrm{bin}(z)|/2) - 1$ if $|\mathrm{bin}(z)|$ is even. Since binary subtraction of 1 and binary division by 2 are both in ZEROSPACE, our machine can check the parity of symbols of $\mathrm{bin}(z)$ and write the correct $\mathrm{bin}(n)$ on a separate worktape without using up space more than logarithmic in $\mathrm{bin}(z)$.

Now our machine must determine whether $2^{2n} \leqslant z \leqslant 2 \cdot 2^{2n} - 1$ or $2 \cdot 2^{2n} \leqslant z \leqslant 3 \cdot 2^{2n} - 1$ or $3 \cdot 2^{2n} \leqslant z \leqslant 4 \cdot 2^{2n} - 1$, and then output $g(\mathrm{bin}(z)) = (\mathrm{bin}(x), \mathrm{bin}(y))$ accordingly. For this, the machine simply checks whether $2 \cdot 2^{2n} \leqslant z$ and whether $3 \cdot 2^{2n} \leqslant z$. Since binary multiplication, the order relation, and raising 2 to a power of n are all in LOGSPACE, the above checking, being a composition of these three functions, can be carried out within space logarithmic in $\mathrm{bin}(z)$.

Finally, to output $g(\mathrm{bin}(z)) = (\mathrm{bin}(x), \mathrm{bin}(y))$, consider the case where $2^{2n} \leqslant z \leqslant 2 \cdot 2^{2n} - 1$. Then $\mathrm{bin}(x) = \lfloor (\mathrm{bin}(z) - 2^{2 \cdot \mathrm{bin}(n)})/2^{\mathrm{bin}(n)} \rfloor$ and $\mathrm{bin}(y) = (\mathrm{bin}(z) - 2^{2 \cdot \mathrm{bin}(n)}) (\mathrm{mod}\ 2^{\mathrm{bin}(n)}) + 2^{\mathrm{bin}(n)}$. We observe that binary division by 2^t and then taking the floor of the result involves simply ignoring the first t symbols of the dividend, and binary $b(\mathrm{mod}\ 2^t)$ is simply the first t symbols of $\mathrm{bin}(b)$. Hence, arguments similar to those in part (a) show that both $\mathrm{bin}(x)$ and $\mathrm{bin}(y)$ can be computed and explicitly written down within space logarithmic in $\mathrm{bin}(z)$. All this also hold for the cases $2 \cdot 2^{2n} \leqslant z \leqslant 3 \cdot 2^{2n} - 1$ and $3 \cdot 2^{2n} \leqslant z \leqslant 4 \cdot 2^{2n} - 1$. And the conversion of $\mathrm{bin}(x)$ and $\mathrm{bin}(y)$ to $\mathrm{tal}(x)$ and $\mathrm{tal}(y)$ is linear in $\mathrm{bin}(z)$, and hence logarithmic in $\mathrm{tal}(z)$.

$\square$

We now proceed to prove that $\mathrm{Bin}(\omega)$ is LOGSPACE set-isomorphic to $\mathrm{B}_k(\omega)$, $k \geqslant 3$, as a corollary to the next few lemmas, specifically, Lemmas 2.3.5–2.3.9. In Lemma 2.3.4 and in Lemma 2.3.5, we in fact prove the existence of an order-isomorphism as opposed to just a set-isomorphism. Although the very next Lemma 2.3.4 is not used to prove that $\mathrm{Bin}(\omega)$ is LOGSPACE set-isomorphic to $\mathrm{B}_k(\omega)$, $k \geqslant 3$, it is interesting in its own right and will prove useful later in Theorem 3.2.12.

Lemma 2.3.4. *The set* $\mathrm{Bin}(\omega) \setminus \{1\}^*$ *is LOGSPACE order-isomorphic to* $\mathrm{Bin}(\omega)$.

Proof: Let $\phi : \text{Bin}(\omega) \setminus \{1\}^* \to \text{Bin}(\omega)$ be defined by $\phi(x) = x + 1 - |x|$. Then $\phi(0) = 0$ and $\phi(2) = 1$. Note that $\phi(1)$ is undefined since $1 \notin \text{Bin}(\omega) \setminus \{1\}^*$.

To show that ϕ is one-to-one and order-preserving, let $x \in \text{Bin}(\omega) \setminus \{1\}^*$ such that $|x| = n > 2$. (The case $|x| \leqslant 2$ is already taken care of by the fact that $\phi(0) = 0$ and $\phi(2) = 1$.) Then $2^{n-1} \leqslant x \leqslant 2^n - 2$. Any two distinct numbers between 2^{n-1} and $2^n - 2$ have the same length $|x| = n$. Hence, all the numbers between 2^{n-1} and $2^n - 2$ get mapped by ϕ to distinct numbers between $2^{n-1} + 1 - n$ and $2^n - 1 - n$, and so ϕ is order-preserving between 2^{n-1} and $2^n - 2$.

To show that ϕ is onto, let $b \in \text{Bin}(\omega)$. Since, as observed above, $\phi(2^{n-1}) = 2^{n-1} + 1 - n$ and $\phi(2^n - 2) = 2^n - 1 - n$, and ϕ is order-preserving between 2^{n-1} and $2^n - 2$, it suffices to show that for every $b \in \text{Bin}(\omega)$, there exists an n such that $2^{n-1} + 1 - n \leqslant b \leqslant 2^n - 1 - n$. We shall show this by induction on $b \geqslant 2$. (For $b = 0, 1$, we already have $\phi(0) = 0$ and $\phi(2) = 1$). For $b = 2$ we have $n = 3$. Now suppose $2^{n-1} + 1 - n \leqslant b \leqslant 2^n - 1 - n$ for some n. If $b < 2^n - 1 - n$, then the same n works for $b + 1$. Otherwise, $b = 2^n - 1 - n$ implies $b + 1 = 2^n - n = 2^{(n+1)-1} + 1 - (n + 1)$, and so $n + 1$ works for $b + 1$.

It remains to show that ϕ and its inverse can be computed in logarithmic space.

Let $g(x, y) = x - y$, $h_1(x) = x + 1$, and $h_2(x) = |x|$ be functions defined on binary numbers. Then $h_2 \in$ LOGSPACE (Lemma 2.1.7) and g and h_1 are in ZEROSPACE (Lemma 2.1.6). Hence by the Generalized Space Composition Lemma, $\phi(x) = g(h_1(x), h_2(x))$ is in LOGSPACE.

The argument for ϕ^{-1} is more complicated. Suppose we are given $b \in \text{Bin}(\omega)$ on the input tape. The machine must output $x \in \text{Bin}(\omega) \setminus \{1\}^*$ such that $\phi(x) = b$. Before we describe our machine's operation, consider for a moment the $x \in \text{Bin}(\omega) \setminus \{1\}^*$ such that $\phi(x) = b$ and suppose $|x| = k$. Since x is not all 1's, we have $|x+1| = k$ also. We claim that $|b| = k$ or $|b| = k - 1$. We have $\phi(x) = x + 1 - k = b$, and hence $|b| \leqslant |x+1| = k$. Now to show that $|b| \geqslant k-1$, it suffices to show that $b = x+1-k \geqslant 2^{k-2}$. Since $x \geqslant 2^{k-1}$, we have $x+1-k \geqslant 2^{k-1}+1-k$. Now if $2^{k-1}+1-k < 2^{k-2}$, then we have $2^{k-1}-2^{k-2} < k-1$, that is, $2 \cdot 2^{k-2}-2^{k-2} < k-1$, and hence $2^{k-2} < k-1$, a contradiction. It follows that $b = x+1-k \geqslant 2^{k-1}+1-k \geqslant 2^{k-2}$, and therefore $k-1 \leqslant |b| \leqslant k$, as we claimed. Now since $\phi(x) = b = x+1-|x|$, and $|b| = k = |x|$ or $|b| = k-1 = |x|-1$, we are left with only two possibilities for the output x: Either $x = b+|b|-1$ or $x = b+|b|$. Suppose it so happens that $x = b+|b|-1$, which will be the case if $|b| = k = |x|$. Then $\phi(x) = b$ implies $x+1-|x| = b$, i.e., $(b+|b|-1)+1-|(b+|b|-1)| = b$, and hence $|b| = |(b+|b|-1)|$.

Thus computing $x = \phi^{-1}(b)$ involves checking whether $|b| = |(b+|b|-1)|$, in which case $x = b+|b|-1$; otherwise $x = b+|b|$. To show that all this can be done without using space more than logarithmic in $|b|$, consider the following functions defined on binary numbers or binary number pairs: $h_1(y) = |y|$, $h_2(y) = y-1$, $h_3(y,z) = y+z$, and h_4 is the order relation on $\mathrm{Bin}(\omega)$, that is, $h_4(y,z) = 1$ (resp. 0) if $y < z$ (resp. $y \geqslant z$). Now h_1 is in LOGSPACE, while the remaining h_i are in ZEROSPACE. It follows from the Generalized Space Composition Lemma that checking whether $|b| = |(b+|b|-1)|$, which is the same as checking whether $N = h_4[h_1(b) - h_1(h_3(h_1(b), h_2(b)))]$ is 1 or 0, is in LOGSPACE. Since $x = h_3(h_2(b), h_1(b))$ if $N = 1$ and $x = h_3(id(b), h_1(b))$ if $N = 0$, the computation of N followed by the outputting of x is in LOGSPACE.

$\square$

Lemma 2.3.5. *The set* $\mathrm{Bin}(\omega)$ *is ZEROSPACE order-isomorphic to the set* $\{0,1\}^*$.

Proof: Let the mapping f from $\{0,1\}^*$ to $\mathrm{Bin}(\omega)$ be defined by $f(\sigma) = (\sigma^\frown 1) - 1$ if $\sigma \neq \emptyset$ and $f(\emptyset) = 0$. We note that since $\sigma^\frown 1 \in \mathrm{Bin}(\omega)$, we have $(\sigma^\frown 1) - 1 \in \mathrm{Bin}(\omega)$, and so f is well-defined.

If $\sigma_1 < \sigma_2$ in the *reverse* lexicographic ordering of $\{0,1\}^*$, then either σ_2 is longer than σ_1 or the first 1 (counting from the right) of σ_2 occurs to the right of the first 1 of σ_1. In either case, we end up with $(\sigma_1\frown 1) < (\sigma_2\frown 1)$ in $\mathrm{Bin}(\omega)$. Hence f is one-to-one and order-preserving. And f is onto because for every binary number b, we have $b = (b+1) - 1$, and if $b \neq 0$, then $b + 1 = \sigma^\frown 1$ for some nonempty string $\sigma \in \{0,1\}^*$.

It remains to show that both f and f^{-1} are in ZEROSPACE.

Given σ on the input tape, our machine outputs $(\sigma^\frown 1) - 1$ without using any space as follows: If σ is the empty string, then the machine outputs 0. Otherwise, the machine composes the ZEROSPACE appending of 1 at the end of the input string followed by the ZEROSPACE subtraction of 1 in binary.

For the other direction, our machine adds 1 in binary but simply does not write the very last symbol on the output tape. This is done as follows: Given a binary number b on the input tape, if $b = 0$, the machine does nothing, and if b is all 1's, the machine ouputs a 0 for each 1 of b. Otherwise, the machine uses special states that allow the input cursor to advance two places to the right from each position i on the input tape, and then return to position i. Once the input cursor returns to position i, the machine outputs the ith symbol of $b + 1$, and stops after outputing a symbol of $b + 1$ only if the input cursor encountered a $\sqcup$ the last time it advanced two places to the right.

$\square$

It is not easy to generalize the above lemma completely by proving the existence of a ZEROSPACE order-isomorphism between $\mathrm{B}_k(\omega)$ and $\{0, 1, \ldots, k-1\}^*$. More specifically, we cannot change the map in the proof of that lemma to the map $\sigma \mapsto (\sigma^\frown(k-1)) - (k-1)$ as this map is not onto $\mathrm{B}_k(\omega)$. However, the existence of a LOGSPACE set-isomorphism between $\mathrm{B}_k(\omega)$ and $\{0, 1, \ldots, k-1\}^*$ is implied by the next three lemmas.

Lemma 2.3.6. *For each $k \geqslant 3$, the set $\mathrm{B}_k(\omega)$ is ZEROSPACE set-isomorphic to the* (k-1)-fold *disjoint union* $\{0, 1, \ldots, k-1\}^* \setminus \{\emptyset\} \oplus \cdots \oplus \{0, 1, \ldots, k-1\}^* \setminus \{\emptyset\}$.

Proof: For each $n \geqslant 1$, let 0^n denote a string of n zeros. Let σ denote a string which has at least one nonzero symbol from the set $\{0, 1, \ldots, k-1\}$. We define a mapping f from $\mathrm{B}_k(\omega)$ to the *(k-1)-fold* disjoint union using the following set of rules:

$$\begin{array}{lll}
0 \to \langle 0, 0\rangle & & \\
1 \to \langle 0, 00\rangle & 0^n{}^\frown 1 \to \langle 0, 0^{n+2}\rangle & \sigma{}^\frown 1 \to \langle 0, \sigma\rangle \\
2 \to \langle 1, 0\rangle & 0^n{}^\frown 2 \to \langle 1, 0^{n+1}\rangle & \sigma{}^\frown 2 \to \langle 1, \sigma\rangle \\
3 \to \langle 2, 0\rangle & 0^n{}^\frown 3 \to \langle 2, 0^{n+1}\rangle & \sigma{}^\frown 3 \to \langle 2, \sigma\rangle \\
\vdots & \vdots & \vdots \\
k-2 \to \langle k-3, 0\rangle & 0^n{}^\frown(k-2) \to \langle k-3, 0^{n+1}\rangle & \sigma{}^\frown(k-2) \to \langle k-3, \sigma\rangle \\
k-1 \to \langle k-2, 0\rangle & 0^n{}^\frown(k-1) \to \langle k-2, 0^{n+1}\rangle & \sigma{}^\frown(k-1) \to \langle k-2, \sigma\rangle
\end{array}$$

This mapping is defined on all of $\mathrm{B}_k(\omega)$ because every nonzero k-ary number ends in one of $0, 1, \ldots, k-1$. Evidently f is one-to-one and onto the disjoint union. It is also evident that computing the f-value of a k-ary number or the f^{-1}-value of an element of the disjoint union does not use up any space because k-ary addition and subtraction are in ZEROSPACE.

$\square$

Lemma 2.3.7. *For each $k \geqslant 3$, the* (k-1)-fold *disjoint union* $\{0, 1, \ldots, k-1\}^* \setminus \{\emptyset\} \oplus \cdots \oplus \{0, 1, \ldots, k-1\}^* \setminus \{\emptyset\}$ *is ZEROSPACE set-isomorphic to the set* $\{0, 1, \ldots, k-1\}^* \setminus \{\emptyset\}$.

Proof: As in the previous proof, we let 0^n denote a string of n zeros for $n \geqslant 1$. Let σ denote a string that has at least one nonzero symbol from the set $\{0, 1, \ldots, k-1\}$ and let τ denote a (possibly empty) string of $\{0, 1, \ldots, k-1\}^*$. We define a mapping from $\{0, 1, \ldots, k-1\}^* \setminus \{\emptyset\}$ to the *(k-1)-fold* disjoint union using the following set of rules:

$$\begin{array}{lll}
2 \to \langle 1, 0\rangle & 2{}^\frown 0^n \to \langle 1, 0^{n+1}\rangle & 2{}^\frown \sigma \to \langle 1, \sigma\rangle \\
3 \to \langle 2, 0\rangle & 3{}^\frown 0^n \to \langle 2, 0^{n+1}\rangle & 3{}^\frown \sigma \to \langle 2, \sigma\rangle \\
\vdots & \vdots & \vdots \\
k-2 \to \langle k-3, 0\rangle & (k-2){}^\frown 0^n \to \langle k-3, 0^{n+1}\rangle & (k-2){}^\frown \sigma \to \langle k-3, \sigma\rangle \\
k-1 \to \langle k-2, 0\rangle & (k-1){}^\frown 0^n \to \langle k-2, 0^{n+1}\rangle & (k-1){}^\frown \sigma \to \langle k-2, \sigma\rangle
\end{array}$$

In order to deal with strings that start with a 0 or a 1, we define the following additional set of rules:

$$1 \to \langle 0, (k-1)^\frown 0\rangle \qquad 1^\frown 0^n \to \langle 0, (k-1)^\frown 0^{n+1}\rangle \qquad 1^\frown \sigma \to \langle 0, (k-1)^\frown \sigma\rangle$$

$$0^n \to \langle 0, 0^n\rangle$$

$$\begin{array}{llll}
0^\frown(k-1)^\frown\tau \to \langle 0, (k-2)^\frown\tau\rangle & & 0^{n+1\frown}(k-1)^\frown\tau \to \langle 0, 0^{n\frown}(k-2)^\frown\tau\rangle \\
0^\frown(k-2)^\frown\tau \to \langle 0, (k-3)^\frown\tau\rangle & & 0^{n+1\frown}(k-2)^\frown\tau \to \langle 0, 0^{n\frown}(k-3)^\frown\tau\rangle \\
\vdots & & \vdots \\
02^\frown\tau \to \langle 0, 1^\frown\tau\rangle & & 0^{n+1\frown}2^\frown\tau \to \langle 0, 0^{n\frown}1^\frown\tau\rangle \\
01^\frown\tau \to \langle 0, 0^\frown(k-1)^\frown\tau\rangle & & 0^{n+1\frown}1^\frown\tau \to \langle 0, 0^{n+1\frown}(k-1)^\frown\tau\rangle
\end{array}$$

The additional rules above now ensure that the mapping is defined on all of the set $\{0, 1, \ldots, k-1\}^* \setminus \{\emptyset\}$. As in the proof of the previous lemma, it is evident that the mapping is both one-to-one and onto, and also in ZEROSPACE.

□

Lemma 2.3.8. *For each $k \geqslant 3$, the set $\{0, 1, \ldots, k-1\}^* \setminus \{\emptyset\}$ is ZEROSPACE set-isomorphic to the set $\{0, 1, \ldots, k-1\}^*$.*

Proof: We can equip the set $D = \{0, 1, \ldots, k-1\}^*$ with the lexiographic ordering (in which case $\emptyset$ will be the smallest element in D). Then we can map every element of D to its immediate successor (relative to the ordering) in $D \setminus \{\emptyset\}$. The successor of $\emptyset$ is 0 and that of the string $(k-1)^n$ (i.e., the string of length $n \geqslant 1$ all of whose symbols are $k-1$) is $0^{n\frown}1$. And the successor of the string $(k-1)^{n\frown}s^\frown\tau$, where $1 \leqslant s \leqslant k-2$ and τ is a (possibly empty) string, is $(k-1)^{n\frown}(s+1)^\frown\tau$. Hence, it is now evident that the successor function from D to $D \setminus \{\emptyset\}$ is a ZEROSPACE bijection.

□

Lemma 2.3.9. *For each $k \geqslant 3$, the set $\{0, 1, \ldots k-1\}^*$ is LOGSPACE set-isomorphic to the set $\{0, 1\}^*$.*

Proof: First let $g : \{0, 1, \ldots, k-1\} \longrightarrow \{0,1\}^*$ be the function defined by $g(0) = 0^{k-1}$ and $g(i) = 0^{i-1}1$ for $1 \leqslant i \leqslant k-1$. We follow the convention that $0^0 1$ denotes 1 so that $g(1)$ is defined.

Now define the function $f : \{0, 1, \ldots, k-1\}^* \longrightarrow \{0,1\}^*$ using g as follows: $f(\emptyset) = \emptyset$, $f(0^n) = 0^n$, $f(\sigma^\frown 0^n) = f(\sigma)^\frown 0^n$, where σ is a string with at least one nonzero symbol, and $f(\sigma_0\sigma_1 \cdots \sigma_{n-1}) = g(\sigma_0)^\frown g(\sigma_1)^\frown \ldots ^\frown g(\sigma_{n-1})$, where the $\sigma_i \in \{0, 1, \ldots, k-1\}$ and at least one σ_i is nonzero.

To prove that f is one-to-one, let $x = x_0x_1 \cdots x_{m-1}$ and $y = y_0y_1 \cdots y_{n-1}$ be distinct strings of $\{0, 1, \ldots, k-1\}^*$. If one of them is $\emptyset$, then certainly their f-values are different. If both x and y have length 1, then their g-values, and hence, their f-values are different. If either x or y is only 0s, then again their f-values are different. Otherwise let $0 \leqslant i \leqslant \min\{m, n\}$, and let the ith symbol be the first symbol where x and y differ. Then $f(x_0 \cdots x_{i-2}) = g(x_0)^\frown \ldots ^\frown g(x_{i-2}) = g(y_0)^\frown \ldots ^\frown g(y_{i-2}) = f(y_0 \cdots y_{i-2})$. But $g(x_{i-1}) \neq g(y_{i-1})$ as one of these strings has a 1 where the other does not. Consequently, we have $f(x) \neq f(y)$ as well.

To prove that f is onto, we first note that $\emptyset$, 0, and 1 all have inverse images under f. Now suppose that every string of $\{0,1\}^*$ of length $\leqslant n$ has an inverse image under f. Let $x = x_0x_1 \cdots x_n \in \{0,1\}^*$. Then $f^{-1}(x_1 \cdots x_n)$ exists by the induction hypothesis. If $x_0 = 1$, then $f^{-1}(x) = 1^\frown f^{-1}(x_1 \cdots x_n)$. But if $x_0 = 0$, then either x is all 0's, in which case x is its own inverse, or the first 1 of x occurs in position i, where $1 < i \leqslant n$. If $i < k-1$, then since $f^{-1}(x_{i+1} \cdots x_n)$ exists by the induction hypothesis, we have $f^{-1}(x) = i^\frown f^{-1}(x_{i+1} \cdots x_n)$. If $i = q(k-1)$, for some $q \geqslant 1$, then $f^{-1}(x) = 0^{q-1\frown}(k-1)^\frown f^{-1}(x_{i+1} \cdots x_n)$. And if $i = q(k-1) + r$ for some $q \geqslant 1$ and $0 < r \leqslant k-1$, then $f^{-1}(x) = 0^{q\frown} r^\frown f^{-1}(x_{i+1} \cdots x_n)$.

Now to prove that f is in LOGSPACE, suppose we are given a string x of $\{0, 1, \ldots, k-1\}^*$ on the input tape. If $x = \emptyset$, the machine does nothing. If x is all zeros, the machine copies x on to the output tape. Otherwise, the machine must determine if x ends in a sequence of 0's and the position where this sequence starts. To that end, the machine adds 1 in binary on a work tape $T1$ each time it reads a symbol of x. Then, if x ends with a 0, the machine adds 1 in binary on a separate work tape $T2$ each time it reads a 0 while it reads x *backwards*, and stops incrementing on $T2$ once it encounters a nonzero symbol of x. After that, the machine subtracts the binary number on $T2$ from the number on $T1$, and stores the result on another work tape $T3$. Now the machine begins reading x from the left. Each time it reads a symbol of x it outputs the g-value of that symbol, subtracts 1 from $T3$, and proceeds to the next symbol of x. When the number on $T3$ becomes 0, the machine copies the rest of x (which will be zeros only provided the content of $T2$ is nonzero) on to the output tape.

Finally, we prove that f^{-1} can be computed in LOGSPACE. The machine employs the work tapes $T1$, $T2$, and $T3$ *exactly* as above. In addition, the machine has $k-1$ special states to detect any occurrence of a string of $k-1$ zeros. Now suppose we are given $x \in \{0,1\}^*$ on the input tape. If $x = \emptyset$, then the machine does nothing. If x is all zeros, then the machine copies x on to the output tape. If x is not all zeros, the procedure is as follows: Each time the machine reads a 1, it subtracts 1 from $T3$, and then writes a 1 on the output tape until, if ever, the machine reads the first 0 of x. As soon as it reads the first 0 of x, the machine switches to the first of the $k-1$ special states. It switches to each of these $k-1$ special states, one after another, each time it reads a 0, until it reads a 1 of x. (We emphasize that the machine always subracts a 1 on $T3$ when it reads a new input symbol). If the last of the $k-1$ special states is reached without encountering a 1, then the machine outputs a 0 (since $g(0) = 0^{k-1}$). Then if the machine still does not encounter a 1, it switches back to the first of the $k-1$ special states and the procedure is repeated. If a 1 is encountered immediately after switching to the last of the $k-1$ special states, then the machine outputs a 1 and proceeds to the next symbol of x. If a 1 is encountered while the machine is still in one of the $k-1$ special states, that is, in some special state i, with $1 \leqslant i \leqslant k$, then the machine outputs i (since $g(i) = 0^{i-1}1$.) Finally, when the content of $T3$ becomes 0, the machine copies the rest of x (which will be zeros only provided the content of $T2$ is nonzero) on to the output tape.

$\square$

Lemma 2.3.10. *For each $k \geqslant 3$, there exists a LOGSPACE bijection $f : \mathrm{Bin}(\omega) \to \mathrm{B}_k(\omega)$. Furthermore, this f has the following property: There exist constants $c_1, c_2 > 0$ such that for every $n \in \omega$, we have $|f(\mathrm{bin}(n))| \leqslant c_1|\mathrm{bin}(n)|$ and $|f^{-1}(\mathrm{b}_k(n))| \leqslant c_2|\mathrm{b}_k(n)|$.*

Proof: The existence of f is an immediate consequence of Lemmas 2.3.5–2.3.9 and the Space Composition Lemmas I and II. As for the property of f in the statement of the current lemma, an examination of the bijections constructed in the proofs of each of the Lemmas 2.3.5–2.3.9 shows immediately that each of these bijections have this property. Since f is the composition of these bijections, it is now evident that f also has this property.

$\square$

The remainder of this section consists of six lemmas, the first four of which deal with certain subsets and combinations of $\mathrm{Tal}(\omega)$ and $\mathrm{Bin}(\omega)$. We first prove a certain boundedness on the lengths of elements of subsets of $\mathrm{Bin}(\omega)$ that are LOGSPACE set-isomorphic to $\mathrm{Tal}(\omega)$. Our proof is the same as the proof of Lemma 2.4 (a) in Cenzer & Remmel [3]. But we reproduce that proof since we shall refer to it in the proof of Lemma 2.3.12, where we show that $\mathrm{Tal}(\omega)$ and $\mathrm{Bin}(\omega)$ are not LOGSPACE set-isomorphic, and also in the proof of Lemma 2.3.13.

Lemma 2.3.11. *Let B be a subset of $\mathrm{Bin}(\omega)$ which is LOGSPACE set-isomorphic to $\mathrm{Tal}(\omega)$, and let $b_0, b_1, b_2 \ldots$ list the elements of B in the standard ordering, first by length and then lexicographically. Then for some j, k and all $n \geqslant 2$, we have $n \leqslant |b_n|^j$ and $|b_n| \leqslant n^k$.*

Proof: Let ϕ be a LOGSPACE set-isomorphism from $\mathrm{Tal}(\omega)$ onto B. By Corollary 2.2.2, there is a k such that $|\phi(1^n)| \leqslant n^k$ for all $n \geqslant 2$. Moreover, we may assume that k is large enough so that $|\phi(0)|$ and $|\phi(1)|$ are both $\leqslant 2^k$. Then, since ϕ is a bijection, there are, for each $n \geqslant 2$, at least $n+1$ distinct elements $\phi(0), \phi(1), \phi(11), \ldots, \phi(1^n)$ of B all having length $\leqslant n^k$. And since the elements $b_0, b_1, b_2 \ldots$ of B are listed in order, all of the elements $b_0, b_1, \ldots, b_n$ have length $\leqslant n^k$.

To prove the other inequality, we first note that 0 and 1 are the only elements of $\mathrm{Bin}(\omega)$ with length $\leqslant 1$, and hence we have $|b_n| \geqslant 2$ for all $n \geqslant 2$. Since ϕ^{-1} is in LOGSPACE, again by Corollary 2.2.2 there is a j such that $|\phi^{-1}(b_n)| \leqslant |b_n|^j$ for all $n \geqslant 2$. And as before, we may assume that $|\phi^{-1}(0)|$ and $|\phi^{-1}(1)|$ are both $\leqslant 2^j$. Then, since ϕ^{-1} is a bijection, there are, for each $n \geqslant 2$, at least $n+1$ distinct elements $\phi^{-1}(b_0), \phi^{-1}(b_1), \ldots, \phi^{-1}(b_n)$ of $\mathrm{Tal}(\omega)$ all having length $\leqslant |b_n|^j$. It follows that all of $0, 1, \ldots, n$ are $\leqslant |b_n|^j$.

□

Lemma 2.3.12. *For any infinite set M of natural numbers,* $\mathrm{tal}(M)$ *and* $\mathrm{bin}(M)$ *are not LOGSPACE set-isomorphic. In particular, the sets* $\mathrm{Tal}(\omega)$ *and* $\mathrm{Bin}(\omega)$ *are not LOGSPACE set-isomorphic.*

Proof: Let $m_0, m_1, \ldots$ list the elements of M in order. Let $\mathrm{tal}(M) = \{a_0, a_1, \ldots\}$, where $a_i = \mathrm{tal}(m_i)$, and let $\mathrm{bin}(M) = \{b_0, b_1, \ldots\}$, where $b_i = \mathrm{bin}(m_i)$.

Suppose $\phi : \mathrm{tal}(M) \to \mathrm{bin}(M)$ is a LOGSPACE bijection. Then by the argument in the second paragraph of the proof of the previous lemma, there is a j such that $|\phi^{-1}(b_n)| \leqslant |b_n|^j$ for all $n \geqslant 2$. The same argument then allows us to conclude that for each $n \geqslant 2$, there are at least $n+1$ distinct elements $\phi^{-1}(b_0), \phi^{-1}(b_1), \ldots, \phi^{-1}(b_n)$ of $\mathrm{tal}(M)$ all having length $\leqslant |b_n|^j$. Hence one of these elements of $\mathrm{tal}(M)$ must be a_n. It follows that $|a_n| \leqslant |b_n|^j$ for all $n \geqslant 2$. Since $a_n = \mathrm{tal}(m_n)$ and $b_n = \mathrm{bin}(m_n)$, we have $m_n \leqslant |\mathrm{bin}(m_n)|^j$ for all $n \geqslant 2$. By Lemma 2.1.2 (a), we have $2^{|\mathrm{bin}(m_n)|-1} < m_n + 1$, and hence $2^{|\mathrm{bin}(m_n)|-1} < |\mathrm{bin}(m_n)|^j + 1$ for all $n \geqslant 2$. This is evidently a contradiction since j is fixed and M is infinite.

□

We now provide a characterization of those LOGSPACE subsets of $\mathrm{Tal}(\omega)$ that are LOGSPACE set-isomorphic to the whole of $\mathrm{Tal}(\omega)$.

Lemma 2.3.13. *Let A be a LOGSPACE subset of* $\mathrm{Tal}(\omega)$, *and let $a_0, a_1, \ldots$ list the elements of A in the standard ordering. Then the following are equivalent :*

(a) *A is LOGSPACE set-isomorphic to* $\mathrm{Tal}(\omega)$.

(b) *For some k and all $n \geqslant 2$, we have $|a_n| \leqslant n^k$.*

(c) *The canonical bijection between* $\mathrm{Tal}(\omega)$ *and A that associates 1^n with a_n, $n \geqslant 0$, is in LOGSPACE.*

Proof: $(a) \Rightarrow (b)$. We simply use the argument in the first paragraph of the proof of Lemma 2.3.11.

$(c) \Rightarrow (a)$. This is immediate since bijections are set-isomorphisms.

$(b) \Rightarrow (c)$. We first note that the map $a_n \mapsto 1^n$ is in LOGSPACE even without (b). To see this, suppose we are given $a \in A$ on the input tape. Our machine writes down $\mathrm{bin}(a)$ on a work tape $W1$ and $\mathrm{bin}(0)$ on another work tape $W2$. It then composes the LINSPACE conversion of the binary number on $W2$ to tally and the testing of whether this tally number is in A. This composition uses up space linear in the length of the contents of $W2$, and hence logarithmic in $|a|$. If the test for membership in A is positive, then the machine writes a 1 on the output tape. Upon completing the above composition, the machine adds 1 in binary on $W2$, subtracts 1 in binary on $W1$, and then repeats the procedure until the content of $W1$ is 0. At this point, if none of the tests for membership in A had been positive, that is, the machine had not written a single 1 on the output tape, the machine outputs 0, signifying $a = a_0$, the very first element of A. Otherwise, the machine writes one more 1 on the output tape.

Now to see that the map $1^n \mapsto a_n$ is in LOGSPACE, assume (b) and suppose we are given 1^n on the input tape. Since $|a_n| \leqslant n^k$ for $n \geqslant 2$, the idea is to keep checking 0, 1, 11, 111, $\ldots, 1^{n^k}$ for membership in A, and to output the nth element of A found. Our machine begins by writing a 0 in binary on a work tape T. Then on separate work tapes, the machine simulates the composition of the following two procedures: (i) The LINSPACE conversion of the binary number t on tape T to the tally number $\mathrm{tal}(t)$, and (ii) The LOGSPACE testing of whether $\mathrm{tal}(t) \in A$. If $\mathrm{tal}(t) \in A$, then the cursor on the input tape moves right, a 1 is added in binary on tape T, and the above simulation of the composition of the two procedures is repeated. However if $\mathrm{tal}(t) \notin A$, then a 1 is added in binary on tape T, but the cursor on the input tape *does not move right.* The simulation of the composition of procedures (i) and (ii) is then carried out on the current content of T. The whole process is repeated until the cursor on the input tape encounters a $\sqcup$. This means that the very last element of A found during the simulation of the composition of (i) and (ii) is the nth one. At this point, the machine keeps subtracting 1 in binary from tape T and writes a 1 on the output tape with each such subtraction, thereby outputing the nth element. Now by the Space Compositon Lemma III, the simulation of the composition of procedures (i) and (ii) uses up space linear in the contents of T. Since $|a_n| \leqslant n^k$, the correct answer would have been written on the output tape no later than when the binary number t on tape T gets incremented to $\mathrm{bin}(n^k)$. We have $|t| \leqslant |\mathrm{bin}(n^k)| = O(\log(n^k)) = O(\log n)$. This means that although the composition of procedures (i) and (ii) uses up space linear in the contents of T, the space used up is logarithmic in the input 1^n.

$\square$

The next lemma deals with the space complexity classes of certain Cartesian products and disjoint unions.

Lemma 2.3.14. *Let A be a nonempty LOGSPACE subset of* $\mathrm{Tal}(\omega)$*. Then we have*

(a) $A \oplus \mathrm{Tal}(\omega)$ *is LOGSPACE set-isomorphic to* $\mathrm{Tal}(\omega)$ *and* $A \oplus \mathrm{Bin}(\omega)$ *is LOGSPACE set-isomorphic to* $\mathrm{Bin}(\omega)$.

(b) $A \times \mathrm{Tal}(\omega)$ *is LOGSPACE set-isomorphic to* $\mathrm{Tal}(\omega)$ *and* $A \times \mathrm{Bin}(\omega)$ *is LOGSPACE set-isomorphic to* $\mathrm{Bin}(\omega)$.

(c) *Both* $\mathrm{Bin}(\omega) \oplus \mathrm{Bin}(\omega)$ *and* $\mathrm{Bin}(\omega) \times \mathrm{Bin}(\omega)$ *are LOGSPACE set-isomorphic to* $\mathrm{Bin}(\omega)$.

(d) *If B is a nonempty finite subset of* $\mathrm{Bin}(\omega)$*, then both* $B \oplus \mathrm{Bin}(\omega)$ *and* $B \times \mathrm{Bin}(\omega)$ *are LOGSPACE set-isomorphic to* $\mathrm{Bin}(\omega)$.

Proof: (a) First let $C = \{2a : a \in A\} \cup \{2n+1 : n \in Tal(\omega)\} \subseteq \mathrm{Tal}(\omega)$ and let $\phi : A \oplus \mathrm{Tal}(\omega) \to C$ be defined by $\phi(\langle 0, a\rangle) = 2a$ and $\phi(\langle 1, \mathrm{tal}(n)\rangle) = 2n+1$. Since the parity of an input string can be checked by a Turing machine using two special states, it is evident that ϕ is a ZEROSPACE bijection. Now C is a LOGSPACE subset of $\mathrm{Tal}(\omega)$ because (i) C contains every odd number, and (ii) given an even number a (in tally) on an input tape, a Turing machine can simulate the composition of the ZEROSPACE halving of a and the LOGSPACE testing of whether $a/2 \in A$ using up only logarithmic space. And since C is a LOGSPACE subset of $\mathrm{Tal}(\omega)$, we can enumerate C in increasing order, yielding $C = \{c_0, c_1, \dots\}$. From the observation that C contains every odd number, we can conclude that $c_n \leqslant 2n+1$. Hence by the previous lemma, C is LOGSPACE set-isomorphic to $\mathrm{Tal}(\omega)$. It now follows that $A \oplus \mathrm{Tal}(\omega)$ is LOGSPACE set-isomorphic to $\mathrm{Tal}(\omega)$ by the Space Composition Lemma III.

Next let $\phi : A \oplus \mathrm{Bin}(\omega) \to [A \oplus \mathrm{Tal}(\omega)] \oplus [\mathrm{Bin}(\omega) \setminus \mathrm{Tal}(\omega)]$ be defined by $\phi(\langle 0, a\rangle) = \langle 0, \langle 0, a\rangle\rangle$, $\phi(\langle 1, \mathrm{bin}(n)\rangle) = \langle 1, \mathrm{bin}(n)\rangle$ if $\mathrm{bin}(n) \notin \{0\} \cup \{1\}^*$, and $\phi(\langle 1, \mathrm{bin}(n)\rangle) = \langle 0, \langle 1, \mathrm{bin}(n)\rangle\rangle$ if $\mathrm{bin}(n) \in \{0\} \cup \{1\}^*$. Evidently ϕ is a ZEROSPACE bijection. By the previous paragraph, $A \oplus \mathrm{Tal}(\omega)$ is LOGSPACE set-isomorphic to $\mathrm{Tal}(\omega)$. Hence $A \oplus \mathrm{Bin}(\omega)$ is LOGSPACE set-isomorphic to $\mathrm{Tal}(\omega) \oplus [\mathrm{Bin}(\omega) \setminus \mathrm{Tal}(\omega)]$

by the Space Composition Lemma III. But $\mathrm{Tal}(\omega)\oplus[\mathrm{Bin}(\omega)\setminus\mathrm{Tal}(\omega)]$ is ZEROSPACE set-isomorphic to $\mathrm{Bin}(\omega)$. Hence $A\oplus\mathrm{Bin}(\omega)$ is LOGSPACE set-isomorphic to $\mathrm{Bin}(\omega)$.

(b) If A has only one element, this is trivial. If A has at least two elements, let a_0 be one of them, and let $\phi : A \times \mathrm{Tal}(\omega) \to [\{a_0\} \times \mathrm{Tal}(\omega)] \oplus [A \setminus \{a_0\} \ \times \mathrm{Tal}(\omega)]$ be defined by $\phi(\langle a_0, \mathrm{tal}(n)\rangle) = \langle 0, \langle a_0, \mathrm{tal}(n)\rangle\rangle$, and $\phi(\langle a, \mathrm{tal}(n)\rangle) = \langle 1, \langle a, \mathrm{tal}(n)\rangle\rangle$ for $a \neq a_0$, $a \in A$. Then ϕ is a ZEROSPACE bijection. Now $\{a_0\} \times \mathrm{Tal}(\omega)$ is evidently ZEROSPACE set-isomorphic to $\mathrm{Tal}(\omega)$. And we claim that $A \setminus \{a_0\} \ \times \mathrm{Tal}(\omega)$ is LOGSPACE set-isomorphic to some LOGSPACE subset D of $\mathrm{Tal}(\omega)$. To see this, let $g : A \setminus \{a_0\} \ \times \mathrm{Tal}(\omega) \to \mathrm{Tal}(\omega)$ be defined by $g(\langle a, \mathrm{tal}(n)\rangle) = [a, \mathrm{tal}(n)]$, where $[\cdot,\cdot]$ is the pairing function of Lemma 2.3.2. By that same lemma, g is in LOGSPACE. Now let D be the range of g. Then D is a LOGSPACE subset of $\mathrm{Tal}(\omega)$ because to check whether $x \in D$, we just check whether the "first component" of $g^{-1}(x)$ is in $A \setminus \{a_0\}$, and both g^{-1} and A are in LOGSPACE, the former by Lemma 2.3.2 (b). Thus $[\{a_0\} \times \mathrm{Tal}(\omega)] \oplus [A \setminus \{a_0\} \ \times \mathrm{Tal}(\omega)]$ is LOGSPACE set-isomorphic to $\mathrm{Tal}(\omega) \oplus D$, which is itself LOGSPACE set-isomorphic to $\mathrm{Tal}(\omega)$ by part (a). It now follows from the Space Composition Lemma III that $A \times \mathrm{Tal}(\omega)$ is LOGSPACE set-isomorphic to $\mathrm{Tal}(\omega)$.

Finally, consider the mapping ϕ from $\mathrm{Tal}(\omega) \times \mathrm{Bin}(\omega)$ to $\mathrm{Bin}(\omega)$ given by: $\langle 1^m, \mathrm{bin}(n)\rangle \mapsto 0^m{}^\frown 1^\frown \mathrm{bin}(n)$ for $n \neq 0$, $\langle 1^m, 0\rangle \mapsto 0^m{}^\frown 1$, $\langle 0, 0\rangle \mapsto 0$, $\langle 0, \mathrm{bin}(n)\rangle \mapsto 1^\frown \mathrm{bin}(n)$, and $\langle 0, 1\rangle \mapsto 1$. Evidently ϕ is a ZEROSPACE bijection. So $A \times \mathrm{Bin}(\omega)$ is ZEROSPACE set-isomorphic to $A \times [\mathrm{Tal}(\omega) \times \mathrm{Bin}(\omega)]$, which in turn is evidently ZEROSPACE set-isomorphic to $[A \times \mathrm{Tal}(\omega)] \times \mathrm{Bin}(\omega)$, which by the previous paragraph is LOGSPACE set-isomorphic to $\mathrm{Tal}(\omega) \times \mathrm{Bin}(\omega)$, which again is ZEROSPACE set-isomorphic to $\mathrm{Bin}(\omega)$ via ϕ. The result now follows from the Space Composition Lemma III.

(c) Define $\phi : \mathrm{Bin}(\omega) \oplus \mathrm{Bin}(\omega) \to \mathrm{Bin}(\omega)$ by $\langle 0, \mathrm{bin}(n)\rangle \mapsto \mathrm{bin}(2n)$ and $\langle 1, \mathrm{bin}(n)\rangle \mapsto \mathrm{bin}(2n{+}1)$. Since binary multiplication and addition are in LOGSPACE, it follows that ϕ is in LOGSPACE. And ϕ^{-1} is in ZEROSPACE. This is because the computations $\text{bin}(2n) \mapsto \text{bin}(n)$ and $\text{bin}(2n+1) \mapsto \text{bin}(n)$, both involve division by 2, which in turn involves the Turing machine simply ignoring the first symbol of the input and copying the rest of the input on to the output tape.

That $\text{Bin}(\omega) \times \text{Bin}(\omega)$ is LOGSPACE set-isomorphic to $\text{Bin}(\omega)$ is simply Lemma 2.3.3.

(d) Evidently there is a ZEROSPACE set-isomorhism ϕ from B to $C = \{\text{bin}(0), \text{bin}(1), \ldots, \text{bin}(N-1)\}$ for some $N \geqslant 1$. Therefore, the set $B \oplus \text{Bin}(\omega)$ is ZEROSPACE set-isomorphic to $C \oplus \text{Bin}(\omega)$. Moreover, $C \oplus \text{Bin}(\omega)$ is evidently ZEROSPACE set-isomorphic to $\text{Bin}(\omega)$ by the isomorphism $\langle 0, \text{bin}(n)\rangle \mapsto \text{bin}(n)$ and $\langle 1, \text{bin}(n)\rangle \mapsto \text{bin}(n+N)$.

Using the set C of the previous paragraph, the set $B \times \text{Bin}(\omega)$ is ZEROSPACE set-isomorphic to $C \times \text{Bin}(\omega)$, which in turn is LOGSPACE set-isomorphic to $\text{Bin}(\omega)$ via the isomorphism $\langle \text{bin}(i), \text{bin}(x)\rangle \to \text{bin}(i) + \text{bin}(N) \cdot \text{bin}(x)$, where $0 \leqslant i \leqslant N-1$.

□

The final two lemmas in this section deal with embedding sets of finite strings into sets of strings of zeros and ones only.

Lemma 2.3.15. *Let Σ be a finite alphabet. There is an embedding η of Σ^* into $\text{Bin}(\omega)$ which is in ZEROSPACE in either direction.*

Proof: We may identify Σ with $\{0, 1, \ldots, n\}$ for some n. Let $\eta(\emptyset) = 0$ and let $\eta(i_1 i_2 \ldots i_k) = 0^{i_1}10^{i_2}1 \ldots 0^{i_k}1$.

To show that η is in ZEROSPACE, suppose we are given a string σ from Σ^* on the input tape. If $\sigma = \emptyset$, our machine outputs 0. Otherwise each time the machine reads a symbol i_j of σ, it switches to a state q_{i_j} and writes i_j zeros on the output tape, followed by a 1.

As for η^{-1}, if our machine is given 0 on its input tape, it does nothing. If given a nonzero $b \in \text{Bin}(\omega)$ on the input tape, the machine writes a 0 on the output tape if the first symbol of b is 1. But if the first symbol of b is 0, then the machine switches to the first of n special states $q_1, \ldots, q_n$. Then each time the machine reads a 0, it switches to the next special state. Then if it reads a 1 while at state q_i, $1 \leqslant i \leqslant n$, the machine writes i on the output tape, swiches to the starting state, and repeats the procedure starting with the next input symbol. But if the machine reads a 0 while in state q_n, then it outputs an error message because the input does not have the correct form.

$\square$

Lemma 2.3.16. *The coding function* $\langle \sigma_0, \sigma_1, \ldots, \sigma_k \rangle_k$ *for* $\sigma_0, \ldots, \sigma_k \in \{0,1\}^*$ *defined by* $\langle \sigma_0, \sigma_1, \ldots, \sigma_k \rangle_k = \eta(\sigma_0\widehat{\ }2 * \sigma_1\widehat{\ }2 * \cdots * \sigma_{k-1}\widehat{\ }2 * \sigma_k)$, *where* η *is the embedding of the previous lemma, is in ZEROSPACE in either direction.*

Proof: Coding is in ZEROSPACE since our machine simply punctuates the computation of each $\eta(\sigma_i)$ by writing $001 = \eta(2)$ on the output tape. Uncoding is in ZEROSPACE because each occurrence of 001 induces the machine to output a 2, while for any other combinations of 0's and 1's, the machine needs to use just two states to count the number of zeros.

$\square$

CHAPTER 3
SPACE COMPLEXITY OF CERTAIN STRUCTURES

3.1 Basic Structural Lemmas

Our first lemma deals with the basic case where there is a LOGSPACE set-isomorphism between two sets, one of which is the universe of a structure.

Lemma 3.1.1. *Suppose that $\mathcal{A}$ is a LOGSPACE structure and ϕ is a LOGSPACE set-isomorphism from A (the universe of $\mathcal{A}$) onto a set B. Then $\mathcal{B}$ is a LOGSPACE structure, where the functions and relations on its universe B are defined to make ϕ an isomorphism of the structures.*

Proof: To show that B is a LOGSPACE set, we observe that $b \in B$ if and only if $\phi^{-1}(b) \in A$. Hence to test whether $b \in B$, we compose the LOGSPACE computation of $\phi^{-1}(b)$ followed by the LOGSPACE testing of whether $\phi^{-1}(b) \in A$. It follows from the Space Composition Lemma I that $B \in$ LOGSPACE.

Now let R^A be an m-ary relation and let f^A be an n-ary function, both defined on A, and with $m, n \geqslant 1$.

To prove that the relation R^B is in LOGSPACE, we make use of the fact that $R^B(b_1, \ldots, b_m) \iff R^A(\phi^{-1}(b_1), \ldots, \phi^{-1}(b_m))$. Let $h_1(b_1, \ldots, b_m) = \phi^{-1}(b_1)$, $h_2(b_1, \ldots, b_m) = \phi^{-1}(b_2)$, $\ldots$, $h_m(b_1, \ldots, b_m) = \phi^{-1}(b_m)$. The machine which computes h_i, ignores all the strings on its m input tapes except the one on its ith input tape, and simulates the machine for ϕ^{-1} on the ith input string. Consequently, the h_i are all in LOGSPACE. We can regard R^A as a boolean function g such that we have $R^A(\phi^{-1}(b_1), \ldots, \phi^{-1}(b_m)) = 1$ (resp. 0) if and only if we have $g(h_1(b_1, \ldots, b_m), \ldots, h_m(b_1, \ldots, b_m)) = 1$ (resp. 0). Since g and each of the h_i are all in LOGSPACE, the result follows from the Generalized Space Composition Lemma.

Finally, the proof that the function f^B is in LOGSPACE depends upon the fact that $f^B(b_1, \ldots, b_n) = \phi(f^A(\phi^{-1}(b_1), \ldots, \phi^{-1}(b_n)))$. If we let $h_i(b_1, \ldots, b_n) = \phi^{-1}(b_i), 1 \leqslant i \leqslant n$, then the argument of the previous paragraph shows that the function $f^A(\phi^{-1}(b_1), \ldots, \phi^{-1}(b_n)) \in$ LOGSPACE. And since ϕ is in LOGSPACE, the Space Composition Lemma I implies $f^B = \phi \circ f^A \in$ LOGSPACE.

□

In the next lemma, we examine the effect of the space complexity of the "tally representation" of a structure on the space complexity of its "k-ary representation," $k \geqslant 2$. We do not include statements like "If $\mathcal{A} \in$ LINSPACE, then $\mathcal{B} \in$ EXSPACE," and "If $\mathcal{A} \in$ EXSPACE, then $\mathcal{B} \in$ DOUBEXSPACE," since they are incorporated in parts (c) and (e) respectively.

Lemma 3.1.2. *Let $\mathcal{M}$ be a structure with universe $M \subseteq \omega$, and let $\mathcal{A} = \mathrm{tal}(\mathcal{M})$ and $\mathcal{B} = \mathrm{b}_k(\mathcal{M})$, where $k \geqslant 2$. Then we have*

(a) *If $\mathcal{A} \in$ LOGSPACE, then $\mathcal{B} \in$ LINSPACE.*

(b) *If $\mathcal{A} \in$ PLOGSPACE, then $\mathcal{B} \in$ PSPACE.*

(c) *If $\mathcal{A} \in$ PSPACE, then $\mathcal{B} \in$ EXSPACE.*

(d) *If $\mathcal{A} \in$ SUPERPSPACE, then $\mathcal{B} \in$ EXPSPACE.*

(e) *If $\mathcal{A} \in$ EXPSPACE, then $\mathcal{B} \in$ DOUBEXSPACE.*

Proof: Let A (resp. B) denote the universe of $\mathcal{A}$ (resp. $\mathcal{B}$). Let R^A (resp. R^B) be an m-ary relation and let f^A (resp. f^B) be an n-ary function, both defined on A (resp. B), and with $m, n \geqslant 1$. Suppose $\mathcal{A} \in$ LOGSPACE.

(a) To test if $b \in B$, it suffices to test if $\mu_k(b) \in A$ (Lemma 2.1.2). Hence given b on the input tape, we compose the LINSPACE conversion of b to $\mu_k(b)$, followed by the LOGSPACE testing of whether $\mu_k(b) \in A$. By the Space Composition Lemma III, it follows that B is a LINSPACE set.

In order to determine the space complexity class of R^B, we imitate the argument for relations in the proof of the previous lemma. We first let $h_i(b_1, \ldots, b_m) =$

$\mu_k(b_i), 1 \leqslant i \leqslant m$, and regard R^A as a boolean valued function g. Now we have $R^B(b_1, \dots, b_m) = 1$ (resp. 0) if and only if $g(h_1(b_1, \dots, b_m), \dots, h_m(b_1, \dots, b_m)) = 1$ (resp. 0). Since the h_i are in LINSPACE and g is in LOGSPACE, the Generalized Space Composition Lemma imples $R^B \in$ LINSPACE.

Finally, to determine the space complexity class of $f^B(b_1, \dots, b_n)$, we imitate the argument for functions in the proof of the previous lemma. Let $h_i(b_1, \dots, b_n) = \mu_k(b_i), 1 \leqslant i \leqslant n$. Then $f^B(b_1, \dots, b_n) = \mu_k^{-1}(f^A(h_1(b_1, \dots, b_n), \dots, h_n(b_1, \dots, b_n)))$. The function $f^A(h_1(b_1, \dots, b_n), \dots, h_n(b_1, \dots, b_n)))$ is in LINSPACE by the argument used in the previous paragraph. And since μ_k^{-1} is in LOGSPACE, the Space Composition Lemma III implies $f^B \in$ LINSPACE.

The proofs of (b)–(e) are similar except that we use the Space Composition Lemmas IV-VII as well as III and the Generalized Space Composition Lemma.

□

In the next four lemmas, we examine the effect of the space complexity of the "k-ary representation," $k \geqslant 2$ of a structure on the space complexity of its "tally representation." Since the proofs of these lemmas are similar, we shall give the proofs of Lemma 3.1.3 and Lemma 3.1.5 only.

Lemma 3.1.3. *Let $\mathcal{M}$ be a structure with universe $M \subseteq \omega$, and let $\mathcal{A} = \mathrm{tal}(\mathcal{M})$ and $\mathcal{B} = \mathrm{b}_k(\mathcal{M})$, where $k \geqslant 2$. Then we have*

(a) *If $\mathcal{B} \in$ LOGSPACE, then $\mathcal{A} \in$ PLOGSPACE.*

(b) *If $\mathcal{B} \in$ LOGSPACE and for all functions $f^{\mathcal{M}}$, we have $|f^{\mathcal{M}}(m_1, \dots, m_n)| \leqslant c(|m_1| + \cdots + |m_n|)$ for some fixed constant c and all but finitely many n-tuples, then $\mathcal{A} \in$ LOGSPACE.*

Proof: As in the proof of the previous lemma, let A (resp. B) denote the universe of $\mathcal{A}$ (resp. $\mathcal{B}$). Let R^A (resp. R^B) be an m-ary relation and let f^A (resp. f^B) be an n-ary function, both defined on A (resp. B), and with $m, n \geqslant 1$. Suppose $\mathcal{B} \in$ LOGSPACE.

(a) This time we begin by considering how to compute f^A. Suppose we are given $a_1, \ldots, a_n$ on n input tapes of a machine M. To compute $f^A(a_1, \ldots, a_n)$, we will not do a formal composition as we did to compute f^B in the proof of the previous lemma. The reason for this is as follows: Since $\mu_k \in$ LINSPACE, the Generalized Space Composition Lemma and the Space Composition Lemmas I and V imply only that the formal composition of μ_k^{-1}, followed by f^B, followed by μ_k, is in PSPACE, and not necessarily in PLOGSPACE.

Instead, we observe that the length of each $b_i = \mu_k^{-1}(a_i)$ is logarithmic in the length of a_i. Thus there is no harm in explicitly writing down the b_i on n distinct work tapes. The machine M then simulates the computation of $b = f^B(b_1, \ldots, b_n)$ which uses up space logarithmic in $r = |b_1| + \cdots + |b_n|$, and hence logarithmic still in the total length $|a| = |a_1| + \cdots + |a_n|$ of the input strings. And by Corollary 2.2.2, there exist nonzero constants c and $k > n$ such that $|b| \leqslant cr^k$. This means that $|b|$ is polynomial in the length of the b_i, and hence polylogarithmic in the length of the a_i. Hence there is also no harm in explicitly writing down b on a separate work tape. Finally, M simulates the computation of $\mu_k(b) = f^A(a_1, \ldots, a_n)$, which uses up space linear in $|b|$, and hence polylogarithmic in $|a|$. We do not concern ourselves with the actual length of $\mu_k(b)$ because $\mu_k(b)$ is produced on the output tape of M in write-only fashion. Thus f^A is in PLOGSPACE.

Now we prove that both A and R^A are in LOGSPACE. Although it does not make a difference in this particular case whether we do formal compositions (as in the proof of the previous lemma) or we occasionally write down certain intermediate output strings completely (as in the previous paragraph), we follow the latter practice because this implies lower space complexity classes for sets and relations in the proofs of the next three lemmas.

To test if $a \in A$, it suffices to test if $\mu_k^{-1}(a) \in B$. So suppose we are given a on the input tape of a machine M. Then M simulates the computation of $b = \mu_k^{-1}(a)$

and explicitly writes b on a work tape. This uses up space logarithmic in $|a|$ and, of course, $|b| \leqslant k \log(|a|)$ for some nonzero constant k. Now M simulates the testing of whether $b \in B$, which uses up space logarithmic in $|b|$. It follows that M operates in LOGSPACE.

To test if $R^A(a_1, \ldots, a_m)$, suppose we are given $a_1, \ldots, a_m$ on m input tapes of some machine M. In the beginning, M simulates the computation of $b_i = \mu_k^{-1}(a_i)$ for $i = 1, \ldots, m$, and explicitly writes down each b_i on m distinct work tapes. This procedure requires exactly these m work tapes and, of course, for each b_i there exists a nonzero constant k_i such that $|b_i| \leqslant k_i \log(|a_i|)$. Now M simulates the testing of whether $R^B(b_1, \ldots, b_m)$, which uses up space logarithmic in $|b| = |b_1| + \cdots + |b_m|$ It follows that R^A is in LOGSPACE.

(b) The arguments for A and R^A are exactly the same as in part (a).

The argument for $f^A(a_1, \ldots, a_n)$ is very similar to that in part (a). Recall that at a certain point in in part (a) we arrive at the following situation: $|b|$ is polynomial in the length of the b_i and hence polylogarithmic in the length of the a_i. But now, owing to the restriction on all functions $f^{\mathcal{M}}$, $|b|$ is in fact *linear* in the length of the b_i, and hence logarithmic in the length of the a_i. Of course while simulating f^B, the machine still uses up only logarithmic space on the tape(s) not used in writing out b, and b is then used to simulate the computation of $\mu_k(b)$, just as in part (a). This last procedure is linear in $|b|$ and hence logarithmic in the a_i. Thus in this case we have $f^A \in$ LOGSPACE.

□

Next, we combine the statements with hypotheses about $\mathcal{B}$ being in LINSPACE and in PSPACE into one lemma for convenience. Also we do not include the statement "If $\mathcal{B} \in$ PLOGSPACE, then $\mathcal{A} \in$ PSPACE," since it is covered by part (a) of the lemma.

Lemma 3.1.4. *Let $\mathcal{M}$ be a structure with universe $M \subseteq \omega$, and let $\mathcal{A} = \mathrm{tal}(\mathcal{M})$ and $\mathcal{B} = \mathrm{b}_k(\mathcal{M})$, where $k \geqslant 2$. Then we have*

(a) *If $\mathcal{B} \in$ LINSPACE, then $\mathcal{A} \in$ PSPACE.*

(b) *If $\mathcal{B} \in$ LINSPACE and for all functions $f^{\mathcal{M}}$, we have $|f^{\mathcal{M}}(m_1, \dots, m_n)| \leqslant (|m_1| + \dots + |m_n|)^c$ for some fixed constant c and all but finitely many n-tuples, then $\mathcal{A} \in$ PLOGSPACE.*

(c) *If $\mathcal{B} \in$ LINSPACE and for all functions $f^{\mathcal{M}}$, we have $|f^{\mathcal{M}}(m_1, \dots, m_n)| \leqslant c(|m_1| + \dots + |m_n|)$ for some fixed constant c and all but finitely many n-tuples, then $\mathcal{A} \in$ LOGSPACE.*

(d) *If $\mathcal{B} \in$ PSPACE, then $\mathcal{A} \in$ SUPERPSPACE.*

(e) *If $\mathcal{B} \in$ PSPACE and for all functions $f^{\mathcal{M}}$, we have the condition $|f^{\mathcal{M}}(m_1, \dots, m_n)| \leqslant 2^{c(|m_1| + \dots + |m_n|)}$ for some fixed constant c and all but finitely many n-tuples, then $\mathcal{A} \in$ PSPACE.*

(f) *If $\mathcal{B} \in$ PSPACE and for all functions $f^{\mathcal{M}}$, we have $|f^{\mathcal{M}}(m_1, \dots, m_n)| \leqslant (|m_1| + \dots + |m_n|)^c$ for some fixed constant c and all but finitely many n-tuples, then $\mathcal{A} \in$ PLOGSPACE.*

The statement "If $\mathcal{B} \in$ SUPERPSPACE, then $\mathcal{A} \in$ EXPSPACE" is covered by part (a) of the next lemma.

Lemma 3.1.5. *Let $\mathcal{M}$ be a structure with universe $M \subseteq \omega$, and let $\mathcal{A} = \mathrm{tal}(\mathcal{M})$ and $\mathcal{B} = \mathrm{b}_k(\mathcal{M})$, where $k \geqslant 2$. Then we have*

(a) *If $\mathcal{B} \in$ EXSPACE, then $\mathcal{A} \in$ EXPSPACE.*

(b) *If $\mathcal{B} \in$ EXSPACE and for all functions $f^{\mathcal{M}}$, we have the additional condition $|f^{\mathcal{M}}(m_1, \dots, m_n)| \leqslant 2^{(|m_1| + \dots + |m_n|)^c}$ for some fixed constant c and all but finitely many n-tuples, then $\mathcal{A} \in$ SUPERPSPACE.*

(c) *If $\mathcal{B} \in$ EXSPACE and for all functions $f^{\mathcal{M}}$, we have the additional condition $|f^{\mathcal{M}}(m_1, \dots, m_n)| \leqslant 2^{c(|m_1| + \dots + |m_n|)}$ for some fixed constant c and all but finitely many n-tuples, then $\mathcal{A} \in$ PSPACE.*

Proof: As before, let A (resp. B) denote the universe of $\mathcal{A}$ (resp. $\mathcal{B}$). Let R^A (resp. R^B) be an m-ary relation and let f^A (resp. f^B) be an n-ary function, both defined on A (resp. B), and with $m, n \geqslant 1$. Suppose $\mathcal{B} \in$ EXSPACE.

(a) To test if $a \in A$, our machine explicitly writes $b = \mu_k^{-1}(a)$, and then simulates the EXSPACE testing of whether $b \in B$. Since $|b| \leqslant k \log(|a|)$ for some nonzero constant k, the testing of whether $b \in B$ takes space $\leqslant 2^{r|b|} \leqslant 2^{rk\log(|a|)} = |a|^{rk}$, where r is some nonzero constant. Thus A is in PSPACE.

To test if $R^A(a_1, \ldots, a_m)$, our machine simulates the computation of $b_i = \mu_k^{-1}(a_i)$ for $i = 1, \ldots, m$, and explicitly writes down each b_i on m distinct work tapes. For each b_i there exists a nonzero constant k_i such that $|b_i| \leqslant k_i \log(|a_i|)$. Let $b = |b_1| + \cdots + |b_m|$ and let $a = |a_1| + \cdots + |a_m|$. Now our machine simulates the EXSPACE testing of whether $R^B(b_1, \ldots, b_m)$, which uses up space $\leqslant 2^{rb} \leqslant 2^{s \log a} = a^s$, where r and s are nonzero constants. It follows that R^A is in PSPACE.

We observe here that since A and R^A are already in PSPACE, no amount of restriction on the lengths of outputs of the functions in $\mathcal{M}$ can force $\mathcal{A}$ to be in LOGSPACE.

To compute $f^A(a_1, \ldots, a_n)$, our machine first explicitly writes down the $b_i = \mu_k^{-1}(a_i)$, $i = 1, \ldots, n$, on n distinct work tapes. Let $r = |b_1| + \cdots + |b_n|$ and let $a = |a_1| + \cdots + |a_n|$. Then there is a nonzero constant k such that $r \leqslant k \log a$. The machine now simulates the EXSPACE computation of $b = f^B(b_1, \ldots, b_n)$. By Corollary 2.2.2, there exist nonzero constants p and q such that $|b| \leqslant 2^{p2^{qr}} \leqslant 2^{p2^{qk \log a}} = 2^{pa^{qk}}$. It follows that f^A is in EXPSPACE.

(b) The arguments for A and R^A are exactly the same as those in part (a).

The argument for $f^A(a_1, \ldots, a_n)$ is very similar to that in part (a). Recall that in part (a) we arrive at the following situation: By Corollary 2.2.2, there exist nonzero constants p and q such that $|b| \leqslant 2^{p2^{qr}} \leqslant 2^{p2^{qk \log a}} = 2^{pa^{qk}}$. But now, owing to the restriction on all functions $f^{\mathcal{M}}$, we have $|b| \leqslant 2^{r^c} \leqslant 2^{(k \log a)^c}$. The computation of $\mu_k(b) = f^A(a_1, \ldots, a_n)$ uses up space linear in $|b|$, and hence linear in $2^{(k \log a)^c}$. It follows that f^A is in SUPERPSPACE.

(c) The proof here is similar to that for part (b).

$\square$

In our final result dealing with the effect of the space complexity of the "k-ary representation" of a structure on the space complexity of its "tally representation," we collect together statements with hypotheses about $\mathcal{B}$ being in EXPSPACE and in DOUBEXSPACE into one lemma for convenience. Note that part (e) implies part (b) in Lemma 3.1.6.

Lemma 3.1.6. *Let $\mathcal{M}$ be a structure with universe $M \subseteq \omega$, and let $\mathcal{A} = \mathrm{tal}(\mathcal{M})$ and $\mathcal{B} = \mathrm{b}_k(\mathcal{M})$, where $k \geqslant 2$. Then we have:*

(a) *If $\mathcal{B} \in$ EXPSPACE, then $\mathcal{A} \in$ EXSUPERPSPACE.*

(b) *If $\mathcal{B} \in$ EXPSPACE and for all functions $f^{\mathcal{M}}$, we have $|f^{\mathcal{M}}(m_1, \ldots, m_n)| \leqslant 2^{2^{c(|m_1|+\cdots+|m_n|)}}$ for some fixed constant c and all but finitely many n-tuples, then $\mathcal{A} \in$ EXPSPACE.*

(c) *If $\mathcal{B} \in$ EXPSPACE and for all functions $f^{\mathcal{M}}$, we have $|f^{\mathcal{M}}(m_1, \ldots, m_n)| \leqslant 2^{(|m_1|+\cdots+|m_n|)^c}$ for some fixed constant c and all but finitely many n-tuples, then $\mathcal{A} \in$ SUPERPSPACE.*

(d) *If $\mathcal{B} \in$ DOUBEXSPACE and for all functions $f^{\mathcal{M}}$, we have $|f^{\mathcal{M}}(m_1, \ldots, m_n)| \leqslant 2^{2^{(|m_1|+\cdots+|m_n|)^c}}$ for some fixed constant c and all but finitely many n-tuples, then $\mathcal{A} \in$ EXSUPERPSPACE.*

(e) *If $\mathcal{B} \in$ DOUBEXSPACE and for all functions $f^{\mathcal{M}}$, we have $|f^{\mathcal{M}}(m_1, \ldots, m_n)| \leqslant 2^{2^{c(|m_1|+\cdots+|m_n|)}}$ for some fixed constant c and all but finitely many n-tuples, then $\mathcal{A} \in$ EXPSPACE.*

The above lemma completes our basic examination of structures with both relations and functions.

3.2 Relational, Functional, and Permutation Structures

We begin by considering relational structures and prove that every recursive relational structure is recursively isomorphic to a LOGSPACE structure. However, we are unable to specify a standard universe for this LOGSPACE structure.

Theorem 3.2.1. *If $\mathcal{A} = (A, \{R_i^A\}_{i \in S}, \{c_i^A\}_{i \in U})$ is a recursive relational structure, then $\mathcal{A}$ is recursively isomorphic to a LOGSPACE structure with universe a subset of* $\mathrm{Bin}(\omega)$ *and to a LOGSPACE structure with universe a subset of* $\mathrm{Tal}(\omega)$.

Proof: We recall that by our definition of recursive structure over an effective language, there is a recursive function s such that for all $i \in S$, the symbol R_i^A is an $s(i)$-ary relation symbol. In addition, there is a recursive function σ such that for all $i \in S$, $\sigma(i)$ is the index of a Turing machine which computes R_i^A.

If A is finite, the result is trivial. So suppose A is infinite. Then there exists a recursive bijection $f : \mathrm{Bin}(\omega) \to A$. We can define a recursive structure $\mathcal{M}$ with universe $\mathrm{Bin}(\omega)$ which is recursively isomorphic to $\mathcal{A}$ by defining the interpretations of the relation symbols and the constant symbols so as to make f an isomorphism of $\mathcal{M}$ onto $\mathcal{A}$. Hence we may assume, without loss of generality, that $A = \mathrm{Bin}(\omega)$. We now proceed to define a LOGSPACE structure $\mathcal{B}$ with universe a subset of $\mathrm{Bin}(\omega)$ such that $\mathcal{B}$ is recursively isomorphic to $\mathcal{A}$.

Let each $a \in A = \mathrm{Bin}(\omega)$ be represented in another binary form by $\psi(a) = 1^{a+1}01^{2^t}$, where t is the time, that is, the number of steps required to carry out the following procedure:

Given the number a in binary as input, test, for each $i \leqslant a$, whether the relation $R_i^A(x_1, \dots, x_{s(i)})$ holds for every single $s(i)$-tuple $(x_1, \dots, x_{s(i)})$ from the $s(i)$-fold product $\{0, 1, \dots, a\}^{s(i)}$ of the set $\{0, 1, \dots, a\} \subset$ $\mathrm{Bin}(\omega)$.

We call this procedure "Relations Checking" for convenience. We observe that Relations Checking is a recursive algorithm that is completely uniform in a because of our definition of recursive structure over an effective language. It now follows that there is a Turing machine that can carry out Relations Checking.

We can now define $\mathcal{B} = (B, \{R_i^{\mathcal{B}}\}_{i \in S}, \{c_i^{\mathcal{B}}\}_{i \in U})$ as follows: Let $B = \{\psi(a) : a \in \text{Bin}(\omega)\}$. For each $i \in S$, let $R_i^{\mathcal{B}}(\psi(a_1), \ldots, \psi(a_{s(i)}))$ be true if and only if $R_i^{\mathcal{A}}(a_1, \ldots, a_{s(i)})$ holds. And for each $i \in U$, let $c_i^{\mathcal{B}} = \psi(c_i^{\mathcal{A}})$. Evidently ψ is a recursive isomorphism from $\mathcal{A}$ onto $\mathcal{B}$. To show that $\mathcal{B}$ is a LOGSPACE structure, we need to check that B is a LOGSPACE set and that each relation $R_i^{\mathcal{B}}$ is in LOGSPACE.

We first show that B is in LOGSPACE. Given $b \in \text{Bin}(\omega)$ on the input tape, our machine uses two special states to check whether b has the form $11 \cdots 1011 \cdots 1$. If b is not of this form, then certainly $b \notin B$. This verification of form uses up no space. If b does have the correct form, the machine proceeds to check whether the terminal segment of 1's of b has length 2^t for some $t \geqslant 0$. The machine does this by using two states to advance the input cursor until it points to the first 1 of the terminal segment of 1's, then adding 1 in binary on a work tape $T1$ each time it reads a 1, and stopping when the input cursor reads a $\sqcup$. This uses up logarithmic space. Now the machine uses one state to check the form of the binary number on $T1$. If this number is not of the form $00 \cdots 01$, that is, if it is not a power of 2, then we have $b \notin B$. However, if $b \in B$, then $b = 1^{a+1}01^{2^t}$, for some numbers a and t. At this point, the machine writes a and t *in binary* on two separate work tapes $T2$ and $T3$, respectively, as follows: Each time the machine reads a 1 in the initial segment of 1's of b starting from the *second* 1, it adds 1 in binary on $T2$. This procedure stops when the machine reads the 0 of b. To compute t, the machine uses the fact that the length of the binary number now on $T1$ is $t + 1$. So it adds 1 in binary on $T3$ each time it reads a symbol on $T1$, starting from the second symbol on $T1$. These procedures use up logarithmic space. Now the machine simulates the machine for Relations Checking on separate tapes with a as input, and subtracts 1 in binary from t on $T3$ each time one step of Relations Checking is completed. We then have

$b \in B$ if and only if Relations Checking finishes in exactly t steps, that is, as soon as the contents of $T3$ become 0. Since Relations Checking is "allowed to run" for only t steps and t is logarithmic in $|b|$, we conclude that B is in LOGSPACE.

Now we show that each relation $R_i^{\mathcal{B}}$ is in LOGSPACE. Let M be a Turing machine with $s(i)$ input tapes, and suppose we are given $b_1, \ldots, b_{s(i)} \in B$ on these input tapes. For $k = 1, 2, \ldots, s(i)$, let a_k and t_k be such that $\mathrm{b}_k = 1^{a_k+1}01^{2^{t_k}}$. Let T be the maximum number of steps required to test whether $R_i^{\mathcal{A}}(x_1, \ldots, x_{s(i)})$ holds when $\{x_1, \ldots, x_{s(i)}\} \subseteq \{0, 1, \ldots, i\} \subset \mathrm{Bin}(\omega)$. In the beginning, M employs the LOGSPACE procedure described in the previous paragraph to write the binary numbers a_k on $s(i)$ separate work tapes. The machine's next operation rests on the fact that $R_i^{\mathcal{B}}(b_1, \ldots, b_{s(i)})$ if and only if $R_i^{\mathcal{A}}(a_1, \ldots, a_{s(i)})$. The machine tests whether $R_i^{\mathcal{A}}(a_1, \ldots, a_{s(i)})$ holds by simply simulating the machine for $R_i^{\mathcal{A}}$ with the a_k as its input. If $\{a_1, \ldots, a_{s(i)}\} \subseteq \{0, 1, \ldots, i\}$, then the number of steps, and therefore, the space required to test whether $R_i^{\mathcal{A}}(a_1, \ldots, a_{s(i)})$ holds is at most the constant T. Otherwise, there is a $j \in \{1, 2, \ldots, s(i)\}$ such that $a_j > i$. In that case, the testing of whether $R_i^{\mathcal{A}}(a_1, \ldots, a_{s(i)})$ holds takes at most t_j steps, and t_j is logarithmic in $|b_j| < |b_1| + \cdots + |b_{s(i)}|$. It now follows that $R_i^{\mathcal{B}}$ is in LOGSPACE.

To finish the proof, we note that $\mathrm{tal}(\mathcal{B})$, which has universe $\{\mathrm{tal}(n) : n \in B\}$, is in LOGSPACE by Lemma 3.1.3 (b), and is recursively isomorphic to $\mathcal{A}$.

□

The above result does not hold for structures with functions, as we prove next.

Theorem 3.2.2. *Let $\mathcal{L}_0$ be the language which has no relation symbols, no constant symbols, and exactly one function symbol which is unary. There is a LINSPACE structure $\mathcal{D} = (D, f^{\mathcal{D}})$ over $\mathcal{L}_0$ which is not recursively isomorphic to any LOGSPACE structure over $\mathcal{L}_0$.*

Proof: Let (E_0, f_0), (E_1, f_1), ... be an effective list of all LOGSPACE structures over $\mathcal{L}_0$, and let ϕ_0, ϕ_1, ... be a list of all one-to-one partial recursive functions. We build our structure $\mathcal{D} = (D, f^{\mathcal{D}})$ so that $D \subset \{1\}^* \subset \mathrm{Tal}(\omega)$. For the rest of this proof, all natural numbers are assumed to be given in tally. We need to ensure in our construction of $\mathcal{D}$ that for each $i, j \in \omega$, the following requirement $R_{i,j}$ is met:

$$R_{i,j} : \phi_j \text{ is not a recursive isomorphism from } \mathcal{D} \text{ onto } (E_i, f_i).$$

We recall that the pairing function $[\cdot,\cdot]$ from $\mathrm{Tal}(\omega) \times \mathrm{Tal}(\omega)$ to $\mathrm{Tal}(\omega)$ defined by $[i, j] = \frac{1}{2}[(i + j)^2 + 3i + j]$ is in LOGSPACE by Lemma 2.3.2 (a). Now define the function $\psi : \mathrm{Tal}(\omega) \times \mathrm{Tal}(\omega) \times \mathrm{Tal}(\omega) \to \mathrm{Tal}(\omega)$ by the recursion $\psi(0, i, j) = 2[i, j]+3$ and $\psi(n + 1, i, j) = 2^{\psi(n,i,j)}$. For each $i, j \in \omega$, let $T_{i,j} = \{\psi(n, i, j) : n \in \omega\}$. And finally define $D = \bigcup_{i,j\in\omega} T_{i,j}$. We observe that D is the disjoint union of the $T_{i,j}$, whose "first elements" are the odd numbers $\geqslant 3$ and whose subsequent elements are obtained by repeated exponentation.

We now prove that $D \in$ LINSPACE. Given $z \in \mathrm{Tal}(\omega)$ on the input tape, our machine first checks if $z = 0$, 1, or 2, in which cases $z \notin D$. But if $z \geqslant 3$, then the machine uses two special states to check the parity of the 1's of z. If z is odd, then $z \in D$. All this uses up no space. Now suppose z is even. The idea is to write down each odd number $\leqslant z$ and $\geqslant 3$, and to repeatedly exponentiate that odd number to see if we obtain z. If we obtain a number greater than z after a certain exponentiation, then we try exponentiating the next odd number to see if we obtain z. In this way, if we exhaust all odd numbers $\leqslant z$ without ever obtaining z by exponentiation, then $z \notin D$. More precisely, the machine moves the input cursor to the extreme-left position and writes 3 (i.e., 111) on a work tape $T1$. Then it computes 2^3, as explained in the proof of Lemma 2.3.1. As it writes each 1 of 2^3 on a work tape $T2$, it advances the input cursor one position to the right. If 2^3 gets written completely on $T2$ and the input cursor reads z but not the final 1 of z, then the machine copies the contents of $T2$ on to another work tape $T3$, erases $T2$, moves the input cursor back to the extreme-left position, and then exponentiates the current number on $T3$, moving the input cursor one position to the right each time it writes a 1 on T2, which will contain the output for the current exponentiation. If 2^3 gets written on $T2$ and the input cursor is on the final 1 of z, then $z \in D$. But if the input cursor encounters a $\sqcup$ while 2^3 is not yet completely written, then the machine adds 2 in tally on tape $T1$ (thus obtaining the next odd number), checks that the odd number on $T1$ is $< z$ by advancing the input and $T1$ cursors simultaneously, and then repeats the exponentiation procedure on the contents of $T1$. The machine halts if it determines at some point that $z \in D$ or if the number on $T1$ becomes bigger than z, in which case $z \notin D$. By Lemma 2.3.1, the above exponentiation procedures are linear in the lengths of the contents of $T1$ and $T3$, and hence linear in $|z|$. Moreover, the content of $T2$ can never be longer than z. It follows that $D \in$ LINSPACE.

We now fix $i, j \in \omega$ and proceed to define the function $f = f^{\mathcal{D}}$ on $T_{i,j} = \{a_0, a_1, \ldots\}$. Recall from the definition of $T_{i,j}$ that $a_n = \psi(n, i, j) = 2^{a_{n-1}}$ for all $n \geqslant 1$, and $a_0 = 2[i, j] + 3$. In order to define $f^{\mathcal{D}}$, we first need to prove that there is an $m \in \{0, 1, \ldots\}$ such that the following computations can be successfully completed in at most a_{m+3} steps:

(1) Start to compute $\phi_j(a_0)$. If this computation converges, then certainly it converges in time $< a_m$ for some $m \geqslant 0$. Let $b_0 = \phi_j(a_0)$.

(2) Check that $b_0 \in E_i$.

(3) Compute the sequence $b_1 = f_i(b_0), b_2 = f_i(b_1), \ldots, b_{m+1} = f_i(b_m)$.

The above computations are completely uniform in a_0. We note that if $\phi_j(a_0) \uparrow$, then the condition $R_{i,j}$ is automatically satisfied. So assume that $b_0 = \phi_j(a_0)$ exists. Now it takes some constant amount c_0 of time to compute b_0 and to check that $b_0 \in E_i$. By Lemma 2.3.15, we may assume that $E_i \subset \text{Bin}(\omega)$. Furthermore, by Corollary 2.2.2, there exists an integer $k > 1$ such that for any $y \in E_i \subset \text{Bin}(\omega)$ with $|y| > 1$, we can compute $f_i(y)$ within $|y|^k$ steps. Let c_1 be the time required to compute $f_i(0)$ and $f_i(1)$, and let $c = c_0 + c_1$. We may assume without loss of generality that $|b_0| > 1$. It now follows that carrying out the computations (1) and (2), and then computing the sequence $b_1 = f_i(b_0), \dots, b_{m+1} = f_i(b_m)$ takes at most T steps, where $T = c + |b_0|^k + (|b_0|^k)^k + \cdots + |b_0|^{k^m}$. We may assume in addition that m is large enough so that $c, m < |b_0|^{k^m}$, $|b_0|^2 < 2^{2^m}$, $k < 2^m$, and $m^2 + m < 2^m$. Then $T = c + |b_0|^k + (|b_0|^k)^k + \cdots + |b_0|^{k^m} < (m+1)|b_0|^{k^m} \leqslant |b_0|^{2k^m} < 2^{2^m \cdot 2^{m^2}} = 2^{2^{m^2+m}} < 2^{2^{2^m}} = \exp_3(m)$. Since $a_0 \geqslant 3$ by the definition of ψ, we have $a_m \geqslant a_0 + m \geqslant m + 3$, and hence $a_{m+3} = 2^{2^{2^{a_m}}} \geqslant 2^{2^{2^{m+3}}} = \exp_3(m+3)$. Thus $T < \exp_3(m) < \exp_3(m+3) \leqslant a_{m+3}$, and so there is an m such that the computations (1), (2), and (3) can be successfully completed within a_{m+3} steps.

It follows from the previous paragraph that there is a *least* $s \geqslant 0$ such that the computations (1), (2), and (3) can be successfully completed within a_s steps. The definition of f on $T_{i,j} = \{a_0, a_1, \dots\}$ now involves considering two cases. For $t \neq s$, we let $f(a_t) = a_{t+1}$. To compute $f(a_s)$, we first let $b_0 = \phi_j(a_0)$ and compute $f_i^{(s+1)}(b_0)$. Then if $f_i^{(s+1)}(b_0) = b_0$, we define $f(a_s) = a_{s+1}$. But if $f_i^{(s+1)}(b_0) \neq b_0$, we define $f(a_s) = a_0$. This ensures that condition $R_{i,j}$ is satisfied.

It remains to prove that f can be computed in linear space. Given $x \in D$ on the input tape, our machine first computes the unique triple (n, i, j) such that $x = \psi(n, i, j)$. This computation uses the fact that n, i, and j are all less than $x (\geqslant 3)$. The machine begins by checking if x is odd. If so, then $n = 0$. Now the machine lists all i and j less than x, starts to compute $2[i, j] + 3$, and moves the input cursor one place to the right each time it writes a 1 of $2[i, j] + 3$, until it writes the correct i and j such that $2[i, j] + 3 = x$. Since the i and j are explicitly written down and the pairing function $[\cdot, \cdot]$, along with tally addition and multiplication, is in LOGSPACE, the above procedures use up linear space. If x is even, the machine carries out the procedure described in the proof that D is in LINSPACE to find the odd number $r < x$ which, after a certain number of exponentiations, gives x. The number n of times r has to be exponentiated to yield x is explicitly written down by adding a 1 with each exponentiation. And the i and j such that $2[i, j] + 3 = r$

are obtained in the manner described in the case where x is odd. Once again, these procedures use up linear space.

As soon as the machine finishes computing the triple (n, i, j) such that $x = \psi(n, i, j)$, it "knows" that $a_0 = r = 2[i, j]+3 = \psi(0, i, j)$ and $x = a_n = r^n = \psi(n, i, j)$. At this point, the machine simulates the recursive computations (1), (2), and (3), while keeping track of the number N of steps carried out. We recall that this is possible for our machine since the computations (1), (2), and (3) are uniform in $a_0 = r$. If these computations are not completed within $a_n = x$ steps, then n is not the "least s" that was used to define $f(a_s)$ above. Hence $f(a_n) = a_{n+1} = 2^{a_n}$, and so the machine outputs $f(x) = 2^x$, which uses up linear space by Lemma 2.3.1. But if these computations are completed within $a_n = x$ steps, i.e., if $N \leqslant x$, then the machine must determine if n is the "least s." The machine does this by computing $a_0(= r)$, $a_1(= 2^{a_0})$, $a_2(= 2^{a_1})$, $\ldots$, until it arrives at the first a_s such that $a_s \geqslant N$. This uses up linear space. If $a_s \neq a_n$, that is, if $s < n$, then once again, $f(a_n) = a_{n+1}$ and the machine outputs $f(x) = 2^x$. But if $a_s = a_n$, then the machine simulates the recursive checking of whether $f_i^{n+1}(b_0) = b_0$. Here $b_0 = \phi_j(a_0) = \phi_j(\psi(0, i, j))$. This checking certainly takes $\leqslant a_s$ steps. Now if $f_i^{n+1}(b_0) = b_0$, then $f(a_n) = a_{n+1}$ as before, and the machine outputs $f(x) = 2^x$. Otherwise $f(a_n) = a_0$ and the machine outputs r. Thus f is in LINSPACE.

□

The function $f^{\mathcal{D}}$ constructed in the proof of the above theorem is a permutation which has infinite orbits and, possibly, finite orbits. Thus we have in fact proved the following:

Theorem 3.2.3. *There is a recursive permutation structure which is not recursively isomorphic to any LOGSPACE permutation structure.*

This result now leads us naturally to a consideration of permutation structures according to the number and size of their orbits. We begin with the simplest case, namely, permutation structures in which all orbits are finite. In this case, we are not able to specify in advance the universe of the LOGSPACE structure that will be proven to be recursively isomorphic to our given permutation structure.

Theorem 3.2.4. *Let (A, f) be a finitary recursive permutation structure. Then (A, f) is recursively isomorphic to a LOGSPACE structure (B, f^B), where B is a LOGSPACE subset of* $\mathrm{Bin}(\omega)$.

Proof: As explained in the proof of Theorem 3.2.1, we may assume that A is infinite and we may further assume that $A = \mathrm{Tal}(\omega)$. For each $a \in A$, let $t(a)$ be the number of steps required to compute $f(a)$ and let $T(a) = \sum_{b \in \mathcal{O}_f(a)} t(b)$. Thus $T(a)$ is the time required to compute the orbit of a. The procedure for computing $T(a)$ is simply to compute and store the successive iterations $f^n(a)$ until one of them comes back to a previous one, while keeping track of and adding the time required to compute each $f^n(a)$. Since all the orbits of f are finite, the function T is recursive and therefore can be computed by a Turing machine. We can now define a recursive function ϕ with domain A by $\phi(a) = (a+1)^\frown 01^{2^{T(a)}}$, and let $B = \{\phi(a) : a \in A\}$.

We now prove that B is a LOGSPACE set. Given a string $\sigma \in \mathrm{Bin}(\omega)$, our machine first checks if σ is of the form $1^{a+1}01^{2^t}$ for some a and t using the procedures described in the proof of Theorem 3.2.1. If σ is not of this form, then certainly $\sigma \notin B$. If σ has the correct form, then the machine writes down a and t, again in the manner described in the proof of Theorem 3.2.1. All these procedures use up logarithmic space. Our machine now simulates the machine for computing the orbit of a for exactly t steps, keeping track of the steps by subtracting 1 from t. If the computation is completed in exactly t steps, then $t = T(a)$ and $\sigma \in B$. If the computation is completed sooner, or is not yet complete after t steps, then $\sigma \notin B$. Since t is logarithmic in $|\sigma|$, we have shown that B is a a LOGSPACE set.

Now we define the permutation f^B on B by $f^B((a+1)^\frown 01^{2^{T(a)}}) = (f(a) + 1)^\frown 01^{2^{T(f(a))}}$. Since $T(f(a)) = T(a)$, no time is involved in the computation of

$01^{2^{T(f(a))}}$. And the time required to compute $f(a)$ is $\leqslant T(a) = O(\log(|(a+1)^\frown 01^{2^{T(a)}}|))$. This shows that f^B is a LOGSPACE function.

Finally, to show that ϕ is an isomorphism from (A, f) to (B, f^B), we observe that $\phi(f(a)) = (f(a)+1)^\frown 01^{2^{T(f(a))}} = f^B((a+1)^\frown 01^{2^{T(a)}}) = f^B(\phi(a))$.

□

We now consider situations when a recursive permutation structure is recursively isomorphic to a LOGSPACE permutation structure with a specified universe such as Tal(ω) or Bin(ω). We in fact obtain results for the more general setting of one-to-one functions from a set into itself. We begin with the special case where there are only finitely many orbits.

Theorem 3.2.5. *Let f be a recursive injection of an infinite recursive set A into itself, with only finitely many orbits. Then (A, f) is recursively isomorphic to a LOGSPACE structure (B, f^B), where B may be taken to be either* Tal(ω) *or* Bin(ω).

Proof: Fix B to be either Tal(ω) or Bin(ω). We observe that since A is infinite, not all the orbits under f are finite. So we shall begin by considering a single infinite orbit O. There are two cases.

Case 1. There is an element a in O which is not in the range of f. Then $O = \{f^n(a) : n \in \omega\}$. So we define the permutation f^B by $f^B(x) = x + 1$ for all $x \in B$. (Here $+$ is the operation appropriate to either Tal(ω) or Bin(ω) depending on our choice of B.) This leads to the recursive isomorphism ϕ from (O, f) onto (B, f^B) defined by $\phi(f^n(a)) = \mathrm{b}_k(n)$, where $n \geqslant 0$, and $k = 1$ if $B = \text{Tal}(\omega)$, while $k = 2$ if $B = \text{Bin}(\omega)$. Since f^B is a ZEROSPACE function, it folows that (O, f) is recursively isomorphic to the ZEROSPACE structure (B, f^B).

Case 2. Every element of O is in the range of f. Let a be an arbitrary element of O. Then $O = \{f^n(a) : n \in \mathbb{Z}\}$. Now we define f^C on the ZEROSPACE set $C = B \oplus B$ by $f^C(\langle 1, x\rangle) = \langle 1, x+1\rangle$, $f^C(\langle 0, 0\rangle) = \langle 1, 0\rangle$, and $f^C(\langle 0, x+1\rangle) = \langle 0, x\rangle$, where $x \geqslant 0$. The function f^C is evidently in ZEROSPACE and hence (C, f^C) is a

ZEROSPACE structure. Now define $\phi : C \to O$ by $\phi(\langle 1,0\rangle) = a$, $\phi(\langle 1,n+1\rangle) = f^n(a)$, and $\phi(\langle 0,n\rangle) = f^{-(n+1)}(a)$, where $n \geqslant 0$. It is easy to check that (C, f^C) is recursively isomorphic to (O, f) via ϕ. By Lemma 2.3.14, C is LOGSPACE set-isomorphic to B, and hence by Lemma 3.1.1, (C, f^C) is LOGSPACE isomorphic to a LOGSPACE structure (B, f^B). It follows that (O, f) is recursively isomorphic to (B, f^B).

To complete the proof, let the finite orbits under f be $O_1, \ldots, O_m$, and let the infinite orbits under f be $O_1^*, \ldots, O_n^*$. For each $i = 1, \ldots, m$, the finite structure (O_i, f) is evidently recursively isomorphic to some ZEROSPACE structure (B_i, f^{B_i}), where B_i is a finite subset of B. And owing to either Case 1 or Case 2, each infinite structure (O_i^*, f), $1 \leqslant i \leqslant n$, is recursively isomorphic to some LOGSPACE structure (B, f_i^B). The disjoint union $B_1 \oplus \cdots \oplus B_m$ is evidently ZEROSPACE set-isomorphic to some finite subset D of B, and so the structure $(B_1 \oplus \cdots \oplus B_m, g)$, where $g = f^{B_i}$ on the ith summand B_i, is LOGSPACE set-isomorphic to a LOGPACE structure (D, f^D) by Lemma 3.1.1. The n-fold disjoint union $B \oplus \cdots \oplus B$ is LOGSPACE set-isomorphic to B by Lemma 2.3.14. Hence the disjoint union of D and the n-fold disjoint union $B \oplus \cdots \oplus B$ is also LOGSPACE set-isomorphic to B by Lemma 2.3.14. It now follows from Lemma 3.1.1 that the structure $(D \oplus B \oplus \cdots \oplus B, h)$, where $h = f^D$ on the first summand D and $g = f_i^B$ on the ith summand B from the following n summands, is LOGSPACE isomorphic to a LOGSPACE structure (B, f^B). Since A is the disjoint union of the orbits of f, the structure (A, f) is recursively isomorphic to the structure $(O_1 \oplus \cdots \oplus O_m \oplus O_1^* \oplus \cdots \oplus O_n^*, \overline{f})$. Here $\overline{f}$ is simply f acting appropriately on the ith summand. It now follows that (A, f) is recursively isomorphic to (B, f^B).

□

The next theorem generalizes Theorem 3.4 by considering the case where an injection has infinitely many orbits.

Theorem 3.2.6. *Let f be a recursive injection of an infinite recursive set A into itself, with at least one but only finitely many infinite orbits. Then (A, f) is recursively isomorphic to a LOGSPACE structure (B, f^B), where B may be taken to be either* $\mathrm{Tal}(\omega)$ *or* $\mathrm{Bin}(\omega)$.

Proof: Fix B to be either $\mathrm{Tal}(\omega)$ or $\mathrm{Bin}(\omega)$. We recall that the finite and infinite parts of A are, respectively, $\mathrm{Fin}_f(A) = \{a \in A : |a|_f < \omega\}$ and $\mathrm{Inf}_f(A) = \{a \in A : |a|_f = \omega\}$. Evidently $\mathrm{Fin}_f(A)$ is an r.e. subset of A. Each infinite orbit under f is r.e., and since only finitely many orbits are infinite, it follows that $\mathrm{Inf}_f(A)$ is a finite union of r.e. sets and is therefore itself r.e. We can now conclude that both $\mathrm{Fin}_f(A)$ and $\mathrm{Inf}_f(A)$ are recursive sets since they are both r.e. and they partition the recursive set A.

Now by Theorem 3.2.4, the structure $(\mathrm{Fin}_f(A), f)$ is recursively isomorphic to some LOGSPACE structure (C, f^C). Moreover, we see in the proof of Theorem 3.2.4 that the elements of C are of the form $1^{m+1}01^{2^n}$ for $m, n \geqslant 0$. Consider the function $\phi : C \to \mathrm{Tal}(\omega) \times \mathrm{Tal}(\omega)$ defined by $\phi(1^{m+1}01^{2^n}) = (1^{m+1}, 1^{2^n})$. Evidently ϕ is in ZEROSPACE. And since C is a LOGSPACE set, the set $\phi(C)$ is a LOGSPACE subset of $\mathrm{Tal}(\omega) \times \mathrm{Tal}(\omega)$. But $\mathrm{Tal}(\omega) \times \mathrm{Tal}(\omega)$ is LOGSPACE set-isomorphic to $\mathrm{Tal}(\omega)$ by Lemma 2.3.14. It follows that $\phi(C)$, and therefore C, is LOGSPACE set-isomorphic to a LOGSPACE subset D of $\mathrm{Tal}(\omega)$. Now Lemma 3.1.1 allows us to conclude that (C, f^C) is recursively isomorphic to some LOGSPACE structure (D, f^D). Hence $(\mathrm{Fin}_f(A), f)$ is also recursively isomorphic to (D, f^D).

As for the structure $(\mathrm{Inf}_f(A), f)$, the previous theorem allows us to conclude that $(\mathrm{Inf}_f(A), f)$ is recursively isomorphic to a LOGSPACE structure (B, g^B).

Now define the structure (E, f^E) by letting $E = D \oplus B$, and by letting $f^E(\langle 0, x\rangle) = \langle 0, f^D(x)\rangle$ and $f^E(\langle 1, x\rangle) = \langle 1, g^B(x)\rangle$. Evidently (E, f^E) is recursively isomorphic to (A, f). Since E is LOGSPACE set-isomorphic to B by Lemma 2.3.14

it now follows from Lemma 3.1.1, that (E, f^E) is LOGSPACE isomorphic to some LOGSPACE structure (B, f^B). Hence (A, f) is recursively isomorphic to (B, f^B).

□

Next we consider what happens when f is a recursive injection from a recursive set A into itself and f has no infinite orbits. This means that (A, f) is in fact a finitary recursive permutation structure and we are in the situation of Theorem 3.2.4. But this time we also assume that all the orbits have the same fixed size, which will allow us to specify in advance the universe A.

Theorem 3.2.7. *Let (A, f) be an infinite recursive permutation structure such that all orbits have the same size q for some $q \in \omega$. Then (A, f) is recursively isomorphic to a LOGSPACE structure (B, f^B), where B may be taken to be either* $\mathrm{Tal}(\omega)$ *or* $\mathrm{Bin}(\omega)$.

Proof: We may assume that $A = \mathrm{Bin}(\omega)$, as explained in the proof of Theorem 3.2.1. We may also assume that $q > 1$. Otherwise, f is the identity function id on A, in which case $(A, f) = (\mathrm{Bin}(\omega), id)$ is of course recursively isomorphic to $(\mathrm{Tal}(\omega), id)$ via the isomorphism that takes each binary number to its tally representation.

Now fix $B = \mathrm{Bin}(\omega)$.

Define the permutation f^{B^q} on $\mathrm{B}_q(\omega)$ by $f^{B_q}(nq+r) = nq+r+1$, if $r+1 < q$, and nq if $r + 1 = q$. Here q and $n, r \geqslant 0$ are the appropriate representations of the natural numbers q, n, and r in $\mathrm{B}_q(\omega)$, and $+$ denotes addition with respect to $\mathrm{B}_q(\omega)$. Thus f^{B_q} maps each q-ary number x to its successor unless x happens to be the immediate predecessor of a multiple of q, in which case x gets mapped to the largest multiple of q smaller than x. As a result, the orbits under f^{B_q} are of the form $nq \to (nq + 1) \to \cdots \to (nq + q - 1) \to nq$, where $n \geqslant 0$.

We claim that f^{B_q} is a ZEROSPACE function. Given $x \in \mathrm{B}_q(\omega)$ on the input tape, our machine first checks if $x + 1$ is divisible by q. This is immediate because x is written in reverse q-ary and the machine simply checks if the first symbol of

x is $q-1$. If this first symbol is not $q-1$, then $x+1$ is not divisible by q, and the machine outputs $x+1$ by using the ZEROSPACE algorithm of Lemma 2.1.1. Otherwise, it uses up zero space to subtract the constant $q-1$ from x and output the result (Lemma 2.1.5). It now follows that $(\mathrm{B}_q(\omega), f^{B_q})$ is a ZEROSPACE structure.

Now we proceed to define a recursive isomorphism from (A, f) to $(\mathrm{B}_q(\omega), f^{B_q})$. First, we recall that $A = \mathrm{Bin}(\omega)$ and define the set I by $i \in I \iff (\forall a < i)(\mathcal{O}_f(a) \neq \mathcal{O}_f(i))$. For each $a \in A$, let $n(a)$ be the number of elements of I strictly less than a, let $i(a)$ be the unique element of $I \cap \mathcal{O}_f(a)$, and let $r(a)$ be the unique r such that $f^r(i(a)) = a$. And now let $\phi : A \to \mathrm{B}_q(\omega)$ be defined by $\phi(a) = n(a)q + r(a)$. We claim that ϕ is a recursive isomorphism from (A, f) to $(\mathrm{B}_q(\omega), f^{B_q})$. To show that ϕ is one-to-one, let $a_1, a_2 \in A$ with $a_1 < a_2$. If a_1 and a_2 are in the same orbit, then $n(a_1) = n(a_2)$ but $r(a_1) \neq r(a_2)$, and hence $\phi(a_1) \neq \phi(a_2)$. If a_1 and a_2 are in different orbits, then $n(a_1) < n(a_2)$, and since $r(a_1)$ and $r(a_2)$ are both $< q$, we have $\phi(a_1) < \phi(a_2)$. To show that ϕ is onto, we observe that since every element of $\mathrm{B}_q(\omega)$ can be written as $lq + m$ for some $l \geqslant 0$ and $0 \leqslant m < q$, it follows that there is an element in the orbit of the $(l+1)$th element of I whose ϕ value is $lq + m$. Now for each $a \in A$, we have $n(f(a)) = n(a)$, and $r(f(a)) = r(a) + 1$ or $r(f(a)) = 0$. If $r(f(a)) = r(a) + 1$, then $\phi(f(a)) = n(f(a))q + r(f(a)) = n(a)q + r(a) + 1 = f^{B_q}(n(a)q + r(a)) = f^{B_q}(\phi(a))$. Similarly, $\phi(f(a)) = f^{B_q}(\phi(a))$ if $r(f(a)) = 0$. Thus ϕ is a recursive isomorphism from (A, f) to $(\mathrm{B}_q(\omega), f^{B_q})$.

To complete the proof for the case $B = \mathrm{Bin}(\omega)$, we observe that by Lemma 2.3.10, there is a LOGSPACE set-isomorphism g from $\mathrm{B}_q(\omega)$ to $B = \mathrm{Bin}(\omega)$. Hence by Lemma 3.1.1, we can conclude that $(\mathrm{B}_q(\omega), f^{B_q})$ is LOGSPACE isomorphic to some LOGSPACE structure $\mathcal{B} = (B, f^B)$. Consequently, (A, f) is recursively isomorphic to $\mathcal{B} = (B, f^B)$.

Finally, note that (A, f) is recursively isomorphic to the structure $\mathrm{tal}(\mathcal{B})$, whose universe is, of course, $\mathrm{Tal}(\omega)$. We now prove that $\mathrm{tal}(\mathcal{B})$ is a LOGSPACE structure. We first observe that Lemma 2.3.10 asserts, in addition to the existence of the LOGSPACE bijection $g : \mathrm{B}_q(\omega) \to B = \mathrm{Bin}(\omega)$ in the previous paragraph, the existence of nonzero constants c_1 and c_2 such that $|g(x)| \leqslant c_1|x|$ and $|g^{-1}(x)| \leqslant c_2|x|$ for every x in the domain of g or g^{-1}, as appropriate. Recall that f^{B_q} is in ZEROSPACE and hence $|f^{B_q}(x)| \leqslant c_3|x|$ for some constant $c_3 > 0$. We claim that f^B also has the property that $|f^B(x)| \leqslant c|x|$ for some constant $c > 0$. To see this, let $x \in B$. Then $f^B(x) = g(f^{B_q}(g^{-1}(x)))$ and so $|f^B(x)| \leqslant c_1c_3c_2|x|$. It now follows from Lemma 3.1.3 (b) that $\mathrm{tal}(\mathcal{B})$ is a LOGSPACE structure.

□

We now generalize the previous result by weakening the assumption that every orbit have the same fixed finite size.

Theorem 3.2.8. *Let (A, f) be a finitary recursive permutation structure such that for some $q \in \omega$, there are infinitely many orbits of size q. Then (A, f) is recursively isomorphic to a LOGSPACE structure (B, f^B), where B may be taken to be either* $\mathrm{Tal}(\omega)$ *or* $\mathrm{Bin}(\omega)$.

Proof: Fix B to be either $\mathrm{Tal}(\omega)$ or $\mathrm{Bin}(\omega)$. Let $C = \{a \in A : |a|_f = q\}$. Now C is a recursive subset of A since to decide if $x \in C$, we need only compute $x, f(x), \ldots, f^q(x)$, and then $x \in C$ if and only if $x, f(x), \ldots, f^{(q-1)}(x)$ are distinct and $x = f^q(x)$. It follows from the previous theorem that $(C, f\restriction_C)$ is recursively isomorphic to some LOGSPACE structure (B, g^B). And it follows from Theorem 3.2.4 and our argument in the proof of Theorem 3.2.6 that $(A \setminus C, f\restriction_{(A\setminus C)})$ is recursively isomorphic to some LOGSPACE structure (E, f^E), where $E \subseteq \mathrm{Tal}(\omega)$. Now let $K = B \oplus E$ and let f^K be defined in the natural way by $f^K(\langle 0, x\rangle) = \langle 0, g^B(x)\rangle$ and $f^K(\langle 1, x\rangle) = \langle 1, f^E(x)\rangle$. Then there is a canonical recursive isomorphism between (A, f) and (K, f^K) which maps C to $\{0\} \times B$ and $A \setminus C$ to $\{1\} \times E$. It now follows from Lemma 2.3.14 that K is LOGSPACE set-isomorphic to B, and then from Lemma 3.1.1 that (K, f^K), and hence (A, f), is recursively isomorphic to a LOGSPACE structure (B, f^B).

$\square$

Corollary 3.2.9. *Every infinite finitary recursive permutation structure (A, f) with a finite upper bound on the size of the orbits of (A, f) is recursively isomorphic to a LOGSPACE structure (B, g), where B may be taken to be either* $\text{Tal}(\omega)$ *or* $\text{Bin}(\omega)$.

Now in order to investigate finitary recursive permutations that do not have infinitely many orbits of any fixed size, we first need to examine the possible spectra of a recursive permutation. We primarily consider monic permutations, which is sufficiently general because of Proposition 3.7 and Theorem 3.8 (b) in Cenzer and Remmel [3]. We quote them next as a theorem.

Theorem 3.2.10. (a) *For every finitary recursive permutation structure (A, f), there is a recursive subset B of A such that f is monic on B and* $\text{Spec}(A, f) = \text{Spec}(B, f)$.
(b) *Any two monic finitary recursive permutation structures with the same spectrum are recursively isomorphic.*

Theorem 3.2.11. *For every nonempty r.e. subset P of $\omega \setminus \{0\}$, there is a monic finitary LOGSPACE permutation f of a LOGSPACE subset A of* $\text{Bin}(\omega)$ *such that* $\text{Spec}(A, f) = P$.

Proof: Since P is r.e., there is a machine M that eventually accepts all elements of P. Thus for every $a \in P$, there is a number $s \geqslant 0$ such that M accepts a is s steps. This means that we can write P as the union of an effective increasing sequence P^s of sets so that it requires time s to check whether $a \in P^s$ for any a and s. Define the set A by $A = \{\langle 1^n, 1^{2^s}, 1^i \rangle : n \in P^{s+1} \setminus P^s \text{ and } i < n\}$. Then A is a LOGSPACE set because we can let the program for M run for exactly $s + 1$ steps to see if M accepts 1^n at the end of those steps. In the proof of Theorem 3.2.1, we explained how

$s+1$ can be written on a worktape without using up space more than logarithmic in the input. Also it requires at most logarithmic space to check if $i < n$. Now define the permutation $f : A \to A$ by $f(\langle 1^n, 1^{2^s}, 1^i \rangle) = \langle 1^n, 1^{2^s}, 1^{i+1} \rangle$ if $i+1 < n$, and $f(\langle 1^n, 1^{2^s}, 1^i \rangle) = \langle 1^n, 1^{2^s}, 1^0 \rangle$ if $i+1 = n$. Evidently f is in LOGSPACE. Moreover, f is evidently monic finitary and we have $\mathrm{Spec}(A, f) = P$.

□

Next we strengthen the previous theorem to ensure that we can specify the universe of A to be either $\mathrm{Tal}(\omega)$ or $\mathrm{Bin}(\omega)$ by assuming that $\mathrm{tal}(P) = \{1^n : n \in P\}$ is a LOGSPACE set.

Theorem 3.2.12. *For every subset P of $\omega \setminus \{0\}$ such that* $\mathrm{tal}(P) \in$ *LOGSPACE, there is a monic finitary LOGSPACE permutation structure (B, f) with* $\mathrm{Spec}(B, f)$ $= P$, *where B may be taken to be either* $\mathrm{Tal}(\omega)$ *or* $\mathrm{Bin}(\omega)$.

Proof: First let $B = \mathrm{Tal}(\omega)$. Let P be enumerated in increasing order as $n_0 < n_1 < \cdots$. We define the permutation $f : B \to B$ in such a way that the orbit of size n_0 is an initial segment of B and, for each $k > 0$, the orbit of size n_k is an initial segment of B minus the orbits of size $< n_k$. Hence if $0 \leqslant n < n_0$, then we define $f(\mathrm{tal}(n)) = \mathrm{tal}(n+1)$ if $n+1 < n_0$, and $f(\mathrm{tal}(n)) = 0$ if $n+1 = n_0$. And if $n = n_0 + n_1 + \cdots + n_{k-1} + i$ for some $k > 0$ and $0 \leqslant i < n_k$, then we define

$$f(\mathrm{tal}(n)) = \begin{cases} \mathrm{tal}(n_0 + n_1 + \cdots + n_{k-1} + i + 1) & \text{if} \quad i+1 < n_k, \\ \mathrm{tal}(n_0 + n_1 + \cdots + n_{k-1}) & \text{if} \quad i+1 = n_k. \end{cases}$$

Evidently the permutation f is finitary monic. Moreover, f is in LOGSPACE. This is because to compute $f(1^n)$, our machine tests whether $1^m \in \mathrm{tal}(P)$ for each $m \leqslant n+1$, then writes every two such 1^m in binary on two worktapes and keeps adding them to eventually obtain $\mathrm{bin}(n_0 + n_1 + \cdots + n_{k-1})$, then computes $\mathrm{bin}(i) = \mathrm{bin}(n) - \mathrm{bin}(n_0 + n_1 + \cdots + n_{k-1})$, and then finally converts to tally the correct output $f(1^n)$ depending on whether $i+1 < n_k$ or $i+1 = n_k$, that is, whether $n+1 \in P$ or $n+1 \notin P$.

It is not evident how to compute the function f in the previous paragraph within logarithmic space in the case $B = \mathrm{Bin}(\omega)$. So we give a more elaborate argument where we partition $\mathrm{Bin}(\omega)$ into ω copies B_n of $\mathrm{Tal}(\omega)$, and partition P into ω LOGSPACE sets P_n. We then define LOGSPACE permutations f_n on B_n with spectrum P_n as in the proof of the previous theorem. Finally, the structures (B_n, f_n) are joined together to make a LOGSPACE structure (B, f^B). Instead of partitioning $\mathrm{Bin}(\omega)$ directly, we partition the more amenable set $\mathrm{Bin}(\omega) \setminus \{1\}^*$, and use the fact (Lemma 2.3.4) proven earlier that $\mathrm{Bin}(\omega)\setminus\{1\}^*$ is LOGSPACE isomorphic to $\mathrm{Bin}(\omega)$.

We partition $B = \mathrm{Bin}(\omega) \setminus \{1\}^*$ by letting $B_0 = \{0\} \cup \{0^{n+1}1^{k+1} : n, k \in \omega\}$, and, for $i \geqslant 1$, by letting $B_i = \{\mathrm{bin}(i)^\frown 0^{n+1}1^{k+1} : n, k \in \omega\}$. Evidently the sets B_i, $i \geqslant 0$, are ZEROSPACE set-isomorphic to $\mathrm{Tal}(\omega) \times \mathrm{Tal}(\omega)$, and hence LOGSPACE set-isomorphic to $\mathrm{Tal}(\omega)$. Let ψ be a LOGSPACE isomorphism from $\mathrm{Tal}(\omega)$ onto B_0.

Now we proceed to partition P. Since $\mathrm{tal}(P) \in$ LOGSPACE, for each $m \in \omega$ we can test $0, 1, \ldots, 1^m$ for membership in $\mathrm{tal}(P)$ within space $O(\log m)$ using binary counters. It follows that the function h such that

$$h(1^m) = \begin{cases} 0 & \text{if } \ 1^m \notin \mathrm{tal}(P), \\ 1^k, \text{ where } k = \mathrm{card}[\mathrm{tal}(P) \cap \{1^r : r \leqslant m\}], & \text{if } \ 1^m \in \mathrm{tal}(P), \end{cases}$$

is in LOGSPACE. Now for each $n \geqslant 0$, we define $P_n = \{m : h(1^m) = \mathrm{tal}(2^r(2n + 1)) \text{ for some } r \geqslant 0\}$. Note that each P_n is infinite and that $P = \bigcup_n P_n$. Since we can factor a number m in tally into an odd number times a power of two within logarithmic space by first converting m into binary, it follows that each P_n is a LOGSPACE set. It also follows that the function θ such that

$$\theta(1^x) = \begin{cases} 0 & \text{if } \ 1^x \notin \mathrm{tal}(P), \\ \langle \mathrm{bin}(n), 1^{r+1} \rangle & \text{if } \ h(1^x) = \mathrm{tal}(2^r(2n+1)) \text{ and } n \geqslant 1, \\ \langle 0, 1^{r+1} \rangle & \text{if } \ h(1^x) = \mathrm{tal}(2^r), \end{cases}$$

is also a LOGSPACE function.

Now that we have partitioned $B = \mathrm{Bin}(\omega)\setminus\{1\}^*$ and P, our idea is to uniformly construct a LOGSPACE monic permutation on B_n with spectrum P_n by making use of the permutation f constructed for the case $B = \mathrm{Tal}(\omega)$ above. First, given $x \in \mathrm{Bin}(\omega) \setminus \{1\}^*$, if $x = 0$, then we compute $\mathrm{tal}(n) = \psi^{-1}(0)$. Otherwise, we have $x = \mathrm{bin}(i)^\frown 0^{s+1}1^{t+1}$ for some $i \geqslant 1$, and $s, t \in \omega$. In this case, we compute $\mathrm{tal}(n) = \psi^{-1}(0^{s+1}1^{t+1})$. Then we compute θ on $0, 1, \ldots, n+1$ and find $n_0 < n_1 < \cdots < n_{k-1} < n_k \leqslant n+1$ (if there are any) such that $\theta(1^{n_j}) = \langle \mathrm{bin}(i), 1^j \rangle$. Now if $1^n = \mathrm{tal}(n_0+\cdots+n_{j-1}+l)$ where $l < n_j$, we let $F(x) = \mathrm{bin}(i)^\frown\psi(\mathrm{tal}(n_0+\cdots+n_{j-1}+l+1))$. If $l = n_j$, then we let$F(x) = \mathrm{bin}(i)^\frown\psi(\mathrm{tal}(n_0 + \cdots + n_{j-1}))$. Since θ, ψ, and ψ^{-1} are in LOGSPACE, it follows that F is a LOGSPACE function. Moreover, the structure $(B_i, F\restriction_{B_i})$ is a finitary monic permutation structure with spectrum P_i. Consequently, (B, F) is a LOGSPACE finitary monic permutation structure with spectrum P. And since $B = \mathrm{Bin}(\omega) \setminus \{1\}^*$ is LOGSPACE set-isomorphic to $\mathrm{Bin}(\omega)$, it follows from Lemma 3.1.1 that (B, F) is isomorphic to a LOGSPACE structure $(\mathrm{Bin}(\omega), f)$.

$\square$

We can now state as a corollary our result about finitary recursive permutation structures that do not have infinitely many orbits of any fixed size. Following that, we state our final positive result about finitary monic recursive permutation structures.

Corollary 3.2.13. *Let Q be an r.e. set with an infinite subset P such that* $\mathrm{tal}(P)$ *is in LOGSPACE. Then any finitary monic recursive permutation structure (A, f) with* $\mathrm{Spec}(A, f) = Q$ *is recursively isomorphic to a LOGSPACE permutation structure (B, f^B), where B may be taken to be either* $\mathrm{Tal}(\omega)$ *or* $\mathrm{Bin}(\omega)$.

Proof: We may assume that $0 \notin P$. Fix $B = \mathrm{Tal}(\omega)$ or $\mathrm{Bin}(\omega)$. By the previous theorem, there is monic finitary LOGSPACE permutation structure (B, g^B) with $\mathrm{Spec}(B, g^B) = P$. Let $C = \{a \in A : |a|_f \in P\}$. Then C is an infinite recursive subset of A. The structures (C, f) and (B, g^B) are recursively isomorphic by Theorem 3.2.10 (b). It now follows from the proofs of Theorem 3.2.4 and Theorem 3.2.6 that $(A \setminus C, f)$ is recursively isomorphic to some LOGSPACE structure (E, f^E), where $E \subseteq \mathrm{Tal}(\omega)$. Evidently (A, f) is recursively isomorphic to the structure $(B \oplus E, g)$, where g is g^B or

f^E as appropriate. Since $B \oplus E$ is LOGSPACE set-isomorphic to B by Lemma 2.3.14, it follows from Lemma 3.1.1 that (A, f) is recursively isomorphic to a LOGSPACE structure (B, f^B).

□

Theorem 3.2.14. *For any r.e. degree d, there is an infinite r.e. subset Q of $\omega \setminus \{0\}$ of degree d such that any monic finitary recursive permutation structure (A, f) with* $\mathrm{Spec}(A, f) = Q$ *is recursively isomorphic to a LOGSPACE permutation structure (B, f^B), where B may be taken to be either* $\mathrm{Tal}(\omega)$ *or* $\mathrm{Bin}(\omega)$.

Proof: Let D be an r.e. set of degree d and let $Q = \{2n + 2 : n \in D\} \cup \{2n + 1 : n \in \omega\}$. Then Q has the same degree as D. Moreover, Q has a subset P, the set of odd numbers, such that $\mathrm{tal}(P)$ is in ZEROSPACE. The result now follows from Corollary 3.2.13.

□

As for negative results, we next state Theorem 3.14 of Cenzer and Remmel [3], which, together with Theorem 3.2.14 above shows that the Turing degree of the set $\mathrm{Spec}(A, f)$ does not determine whether a finitary monic recursive permutation structure (A, f) is recursively isomorphic to LOGSPACE permutation over $\mathrm{Tal}(\omega)$ or $\mathrm{Bin}(\omega)$.

Theorem 3.2.15. *For any r.e. degree d, there is a set P of degree d such that no finitary monic recursive permutation structure (A, f) with* $\mathrm{Spec}(A, f) = P$ *can be isomorphic to any primitive recursive permutation structure* $(\mathrm{Tal}(\omega), g)$ *or* $(\mathrm{Bin}(\omega), h)$.

Finally, we conclude our investigation of permutation structures according to the number and size of the orbits by stating a negative result about permutation structures with infinitely many infinite orbits. This result is an immediate consequence of Theorem 3.17 in Cenzer and Remmel [3].

Theorem 3.2.16. (a) *There is a recursive permutation f of a recursive set A with infinitely many orbits, all of type $\mathbb{Z}$, such that (A, f) cannot be recursively embedded in any LOGSPACE structure.*

(b) *There is a recursive injection f of a recursive set A with infinitely many orbits, all of type ω, such that (A, f) cannot be recursively embedded in any LOGSPACE structure.*

3.3 Abelian Groups

We now begin our investigation of LOGSPACE Abelian groups. The results here parallel those for permutation structures. The next theorem is an immediate consequence of Theorem 4.1 in Cenzer and Remmel [3].

Theorem 3.3.1. *There is a recursive Abelian group that is not recursively isomorphic to any LOGSPACE Abelian group.*

Following standard notation in Algebra, we let $\mathbb{Z}$ denote the group of integers with the usual addition. For each natural number $n > 1$, we let $\mathbb{Z}_n$ denote the cyclic group of order n. And for each prime number p, we let $\mathbb{Z}(p^\infty)$ denote the group of rational numbers with denominator a power of p and addition modulo 1. Note that $\mathbb{Z}(p^\infty)$ is a subgroup of the group $\mathbb{Q}/\mathbb{Z}$, the group of rationals modulo 1. We explain the structure of $\mathbb{Z}(p^\infty)$ and its operation more fully in the proof of Lemma 3.3.3. And finally, we denote the additive group of rational numbers by $\mathbb{Q}$. We now define the product of a sequence of groups which is equivalent to the definition of the external weak product of groups in Algebra.

Definition. For any sequence $\mathcal{A}_0, \mathcal{A}_1, \ldots$ of groups, where each $\mathcal{A}_i = (A_i, +_i, -_i, e_i)$, and each $A_i \subseteq \{0,1\}^*$, the *direct product* $\mathcal{A} = \bigoplus_n \mathcal{A}_n$ is the group defined to have domain $A = \{\langle \sigma_0, \sigma_1, \ldots, \sigma_k \rangle : k \in \omega, \sigma_i \in A_i \text{ for } 0 \leqslant i \leqslant k \text{ and } \sigma_k \neq e_k\}$, identity $e^A = \emptyset$, and group operations addition $+^A$ and subtraction $-^A$ defined as follows:

For $\sigma = \langle \sigma_0, \sigma_1, \dots, \sigma_m \rangle$ and $\tau = \langle \tau_0, \tau_1, \dots, \tau_n \rangle$, we define $\sigma +^A / -^A \tau = \rho = \langle \rho_0, \rho_1, \dots, \rho_k \rangle$, where $k = \max\{i : [(i \leqslant m) \wedge (i \leqslant n) \wedge (\sigma_i +_i / -_i \tau_i \neq e_i)] \vee [m < i \leqslant n] \vee [n < i \leqslant m]\}$ and, for $i \leqslant k$,

$$\rho_i = \begin{cases} \sigma_i +_i / -_i \tau_i & \text{for } i \leqslant \min(m, n), \\ \sigma_i & \text{for } n < i \leqslant k, \\ \tau_i & \text{for } m < i \leqslant k. \end{cases}$$

In particular, we let $\bigoplus_\omega \mathcal{G}$ denote the direct product of a countably infinite number of copies of the group $\mathcal{G}$.

Definition. Let B be either $\mathrm{Tal}(\omega)$ or $\mathrm{Bin}(\omega)$. We say that the sequence $\mathcal{A}_0, \mathcal{A}_1, \dots$ of groups, where $\mathcal{A}_n = (A_n, +_n, -_n, e_n)$, is *fully uniformly LOGSPACE over B* if the following hold:

(i) The set $\{\langle b(n), a \rangle : a \in A_n\}$ is a LOGSPACE subset of $B \times B$, where $b(n) = \mathrm{tal}(n)$ if $B = \mathrm{Tal}(\omega)$ and $b(n) = \mathrm{bin}(n)$ if $B = \mathrm{Bin}(\omega)$.

(ii) The functions $F(b(n), a, b) = a +_n b$ and $G(b(n), a, b) = a -_n b$ are both the restrictions of LOGSPACE functions from B^3 to B, where we set $F(b(n), a, b) = G(b(n), a, b) = \emptyset$ if either a or b is not in A_n.

(iii) The function from $\mathrm{Tal}(\omega)$ into B defined by $e(\mathrm{tal}(i)) = e_i$ is in LOGSPACE.

Lemma 3.3.2. *Let B be either* $\mathrm{Tal}(\omega)$ *or* $\mathrm{Bin}(\omega)$. *Suppose that the sequence $\mathcal{A}_i = (A_i, +_i, -_i, e_i)$ of groups is fully uniformly LOGSPACE over B. Then we have*

(a) *The direct product $\mathcal{A}$ of the sequence $\mathcal{A}_i$ is recursively isomorphic to a LOGSPACE group with universe contained in* $\mathrm{Bin}(\omega)$.

(b) *If A_i is a subgroup of A_{i+1} for all i,and if there is a LOGSPACE function $f : \{0,1\}^* \to B$ such that for all $a \in \bigcup_i A_i$, we have $a \in A_{f(a)}$, then the union $\bigcup_i \mathcal{A}_i$ is a LOGSPACE group with universe contained in B.*

(c) *If the sequence is finite, one of the components has universe B and the remaining components have universes that are LOGSPACE subsets of* $\mathrm{Tal}(\omega)$, *then the direct product is recursively isomorphic to a LOGSPACE group with universe B.*

(d) *If the sequence is infinite and if each component has universe B, then the direct* *product is recursively isomorphic to a LOGSPACE group with universe* Bin(ω).

(e) *If each component has universe* Tal(ω) *and there is a uniform constant c such that for each i and any $a, b \in A_i$, we have both $|a +_i b| \leqslant c(|a| +_i |b|)$ and $|a -_i b| \leqslant c(|a| +_i |b|)$, then the direct product is recursively isomorphic to a LOGSPACE group with universe* Tal(ω).

Proof: (a) The domain of $\mathcal{A}$ is recursively isomorphic to $A = \bigcup_{k\in\omega} Q_k$, where $Q_k = \{\langle\sigma_0, \sigma_1, \ldots, \sigma_k\rangle_k : \sigma_i \in A_i \text{ and } \sigma_k \neq e_k\}$. Recall that $\langle\sigma_0, \sigma_1, \ldots, \sigma_k\rangle_k = \eta(\sigma_0^\frown 2 * \sigma_1^\frown 2 * \cdots * \sigma_{k-1}^\frown 2 * \sigma_k)$, where $\eta(i_1 i_2 \ldots i_k) = 0^{i_1}10^{i_2}1\ldots 0^{i_k}1$ (Lemma 2.3.16). Thus $A \subseteq$ Bin(ω). Since each $\sigma_i \in A_i \subseteq \{0,1\}^*$, the string $\eta(\sigma_i)$ cannot have a 001 $= \eta(2)$. Thus the testing that potential elements of A have the correct form requires zero space.

Now suppose we have an element $\eta(\sigma_0) * 001 * \eta(\sigma_1) * 001 * \cdots * 001 * \eta(\sigma_k)$ of Q_k on the input tape. For each $0 \leqslant i \leqslant k$, our machine writes bin$(|\eta(\sigma_i)|)$ on a counter tape. This is done using the occurrences of 001 on the input tape. The machine uses bin$(|\eta(\sigma_i)|)$ to read the symbols of $\eta(\sigma_i)$ only and test whether $\eta^{-1}(\eta(\sigma_i)) \in A_i$, and, also uses bin$(|\eta(\sigma_k)|)$ to check whether $\eta^{-1}(\eta(\sigma_k)) \neq e_k$. Since η is in ZEROSPACE and the sequence $\mathcal{A}_i$ of groups is uniformly LOGSPACE over B, these procedures can be carried out within logarithmic space.

Now we verify that the operations $+^A$ and $-^A$ are in LOGSPACE. Given $\sigma = \langle\sigma_0, \sigma_1, \ldots, \sigma_m\rangle_m$ and $\tau = \langle\tau_0, \tau_1, \ldots, \tau_n\rangle_n$, on two input tapes, our machine first checks the number of occurrences of $\eta(2) = 001$ to see if $m \geqslant n$. This uses up zero space. If $m = n$, then our machine checks whether $\sigma_m +_m / -_m \tau_m = e_m$, whether $\sigma_{m-1} +_{m-1} / -_{m-1} \tau_{m-1} = e_{m-1}$, and so on, using a binary counter as in the previous paragraph. If $\sigma_i +_i / -_i \tau_i = e_i$ for all $0 \leqslant i \leqslant m = n$, then the machine outputs $\emptyset$. Otherwise, suppose $i \in \{0, 1, \ldots, m\}$ is the largest such that $\sigma_i +_i / -_i \tau_i \neq e_i$. Again, using binary counters and by virtue of uniformity of the sequence $\mathcal{A}_i$, the machine outputs $\eta[\eta^{-1}(\eta(\sigma_0)) +_0 / -_0 \eta^{-1}(\eta(\tau_0))] * 001 * \cdots * 001 * \eta[\eta^{-1}(\eta(\sigma_i)) +_i / -_i$

$\eta^{-1}(\eta(\tau_i))]$, using up at most logarithmic space. If $m > n$, then the machine outputs $\eta[\eta^{-1}(\eta(\sigma_0)) +_0 / -_0 \eta^{-1}(\eta(\tau_0))] * 001 * \cdots * 001 * \eta[\eta^{-1}(\eta(\sigma_n)) +_n / -_n \eta^{-1}(\eta(\tau_n))] * \sigma_{n+1} * 001 \cdots * 001 * \sigma_m$, and similarly if $m < n$.

(b) We certainly have $\bigcup_i A_i \subseteq B$. Given $a \in \{0,1\}^*$ on the input tape, to test whether $a \in A = \bigcup_i A_i$, it suffices to compose the LOGSPACE computation of $f(a)$ and the verification that $\langle b(f(a)), a\rangle \in \{\langle b(f(a)), a\rangle : a \in A_{f(a)}\}$. This composition is in LOGSPACE because of uniformity, and the fact that the conversion of $\text{bin}(f(a))$ to $\text{tal}(f(a))$ is linear in $|\text{bin}(f(a))|$ and hence logarithmic in $|a|$.

Our machine can carry out $+^A/-^A$ in LOGSPACE as follows: Given inputs a and b, it uses LOGSPACE compositions to compute and write $\text{bin}(i)$, where $i = \max\{f(a), f(b)\}$. It then outputs $a +_i / -_i b$, which requires no more than logarithmic space because of uniformity.

(c) We may assume without loss of generality that $A_0 = B$. Then for the finite sequence $A_0, A_1, \ldots, A_n$, the universe of the direct product is evidently recursively isomorphic to $A_0 \times A_1 \times \cdots \times A_n$, which is LOGSPACE isomorphic to B by Lemma 2.3.14 (b). Since the direct product is ZEROSPACE isomorphic to a LOGSPACE group by part (a), we can now apply Lemma 3.1.1 to reach our conclusion.

(d) First suppose $B = \text{Tal}(\omega)$. Then the domain of the direct product is recursively isomorphic to $A = \{\langle \text{tal}(n_0), \text{tal}(n_1), \ldots, \text{tal}(n_k)\rangle_k : k \in \omega,\ \text{tal}(n_i) \in A_i$, and $\text{tal}(n_k) \neq e_k\}$. The mapping from A to $\text{Bin}(\omega)$ given by $\langle \text{tal}(n_0), \text{tal}(n_1), \ldots, \text{tal}(n_k)\rangle_k \to \text{tal}(n_0)^\frown 0 * \text{tal}(n_1)^\frown 0 * \cdots * 0 * \text{tal}(n_k)$ is a ZEROSPACE bijection. Hence Lemma 3.1.1 now applies.

Now suppose $B = \text{Bin}(\omega)$. Then the domain of the direct product is $A = \{\langle \sigma_0, \sigma_1, \ldots, \sigma_k\rangle : k \in \omega,\ \sigma_i \in \text{Bin}(\omega)$, and $\sigma_k \neq e_k\}$. For each $\sigma \in \text{Bin}(\omega)$, let σ^- be the result of deleting the 1 at the end of the string $\sigma + 1$. Then the mapping between A and $\{0,1,2\}^*$ given by $\langle \sigma_0, \sigma_1, \ldots, \sigma_k\rangle \to \sigma_0^- 2 \sigma_1^- 2 \cdots 2 \sigma_k^-$ is a ZEROSPACE bijection. By Lemma 2.3.5, Lemma 2.3.9, and the Space Composition

Lemma I, the set $\{0, 1, 2\}^*$ is LOGSPACE set-isomorphic to $\mathrm{Bin}(\omega)$. It follows that A is LOGSPACE set-isomorphic to $\mathrm{Bin}(\omega)$. Once again, Lemma 3.1.1 now applies.

(e) If the sequence is finite, then this is part (c). Otherwise, by part (d), the direct product is recursively isomorphic to a LOGSPACE group $\mathcal{A}$ with universe $\mathrm{Bin}(\omega)$. Since $|a +_i / -_i b| \leqslant c(|a| +_i |b|)$, we have $|a + / -_A b| \leqslant c(|a| +_A |b|)$ also. Lemma 3.1.3 now implies our result.

□

Lemma 3.3.3. *Each of the groups* $\mathbb{Z}$, $\bigoplus_\omega \mathbb{Z}_k$, $\mathbb{Z}(p^\infty)$, *and* $\mathbb{Q}$ *are recursively isomorphic to LOGSPACE groups* (a) *with universe* $\mathrm{Bin}(\omega)$, *and* (b) *with universe* $\mathrm{Tal}(\omega)$.

Proof: The mapping $f : \mathbb{Z} \to \omega$ given by $f(n) = 2n$ if $n \geqslant 0$, and $f(n) = 2(-n) - 1$ if $n < 0$, is a recursive bijection. Let $\oplus$ and $\ominus$ denote, respectively, the corresponding addition and subtraction operations on ω; and let $+$ and $-$ denote the addition and subtraction in $\mathbb{Z}$. Suppose x, $y \in \omega$. If x and y are both even, that is, $x = 2m$ and $y = 2n$ for m, $n \in \omega$, then we must have $x \oplus y = f(m+n) = 2(m+n) = x + y$. And $x \ominus y = f(m-n) = 2(m-n) = x - y$ if $m - n \geqslant 0$, while $f(m-n) = 2(n-m) - 1 = y - x - 1$ if $m - n < 0$. Hence $x \ominus y = x - y$ if $m \geqslant n$, and $x \ominus y = y - x - 1$ if $m < n$. Similarly, if x is even and y is odd, that is, $x = 2m$ and $y = 2n - 1$ for m, $n \in \omega$, $n \geqslant 1$, then $x \ominus y = x + y + 1$. And we have $x \oplus y = x - y - 1$ if $m \geqslant n$, while $x \oplus y = y - x$ if $m < n$. The case where x is odd and y is even is completely symmetric. And finally, if x and y are both odd, that is, $x = 2m - 1$ and $y = 2n - 1$, for m, $n \geqslant 1$, then we have $x \oplus y = x + y + 1$. And $x \ominus y = y - x$ if $m \leqslant n$, while $x \ominus y = x - y - 1$ if $n < m$.

Whether we work with the binary or the tally representation, finding the m and n corresponding to the inputs x and y requires only zero space. Then comparing m and n uses up logarithmic space, and finally outputing $x \oplus y$ and $x \ominus y$ requires logarithmic space. Hence by the Space Composition Lemma I, the group $(\omega, \oplus, \ominus, 0)$, with universe either $\mathrm{Tal}(\omega)$ or $\mathrm{Bin}(\omega)$, is a LOGSPACE group.

The mapping $f : \bigoplus_\omega \mathbb{Z}_k \to B_k(\omega)$ defined by $\langle \sigma_0, \sigma_1, \ldots, \sigma_n \rangle \mapsto \sigma_0\sigma_1 \cdots \sigma_n$, where $\sigma_i \in \mathbb{Z}_k$ and $\sigma_n \neq 0$, is a recursive bijection. The corresponding addition and subtraction operations $\oplus$ and $\ominus$ in $B_k(\omega)$ are coordinatewise and modulo k, but without any carrying involved. Thus the group $\mathcal{G} = (B_k(\omega), \oplus, \ominus, 0)$, is a ZEROSPACE group with the property that $|a \;\oplus / \ominus\; b| \leqslant \max(|a|, |b|)$. By Lemma 2.3.10 and Lemma 3.1.1, the group $\mathcal{G}$ is recursively isomorphic to a LOGSPACE group $\mathcal{H}$ with universe $\mathrm{Bin}(\omega)$ and with the property that $|a \;+^H / -^H\; b| \leqslant c(|a| +^H |b|)$, for some constant c. Hence by Lemma 3.1.3, the group $\mathcal{H}$ is also recursively isomorphic to a group with universe $\mathrm{Tal}(\omega)$.

As for $\mathbb{Z}(p^\infty)$, we will define a group $\mathcal{G}(p^\infty)$ that represents the group $\mathbb{Z}(p^\infty)$. Then we will show that $\mathcal{G}(p^\infty)$ is recursively isomorphic to a LOGSPACE group with universe $\mathrm{Bin}(\omega)$ and to a LOGSPACE group with universe $\mathrm{Tal}(\omega)$. But first we need to recall a few basic facts about the group $\mathbb{Z}(p^\infty)$. This group's underlying set is the set $\{[a/b] \in \mathbb{Q}/\mathbb{Z} : a, b \in \mathbb{Z} \text{ and } b = p^i \text{ for some } i \geqslant 0\}$ of equivalence classes of the congruence relation modulo 1, that is, the relation where $a \in \mathbb{Q}$ is related to $b \in \mathbb{Q}$ if and only if $a - b$ is an integer. So $\mathbb{Z}(p^\infty)$ is generated by the set $\{[0]\} \cup \{[1/p^n] : n \in \mathbb{N}\}$. We have $[0] = [1] = [-1] = [2] = [-2] = \cdots = \mathbb{Z}$, and $[1/p^n] = \{(kp^n + 1)/p^n : k \in \mathbb{Z}\}$. Moreover, $[1/p^n]$ generates the elements $[2/p^n] = \{(kp^n + 2)/p^n : k \in \mathbb{Z}\}$, and $[3/p^n] = \{(kp^n + 3)/p^n : k \in \mathbb{Z}\}$, and so on, including $[(p^n - 1)/p^n] = \{(kp^n + p^n - 1)/p^n : k \in \mathbb{Z}\}$. The addition and subtraction of equivalence classes $[a], [b] \in \mathbb{Z}(p^\infty)$ are defined as follows: $[a] + / - [b] = [a + / - b]$. Note that for $[x/p^n]$, where $1 \leqslant x \leqslant p^n - 1$, we have $-[x/p^n] = [-x/p^n] = [(p^n - x)/p^n]$. Also note that if $x > p^n$, say, $x = p^n + 3$, then $[x/p^n] = [(p^n + 3)/p^n] = [1 + (3/p^n)] = [3/p^n]$.

We will now proceed to define $\mathcal{G}(p^\infty)$. Let the string $e_0e_1 \cdots e_{n-1}$, where each $e_i \in B_p(\omega)$ and $e_{n-1} \neq 0$, represent the element (more precisely, equivalence class) $\left[\frac{e_0}{p} + \frac{e_1}{p^2} + \cdots + \frac{e_{n-1}}{p^n}\right]$ of $\mathbb{Z}(p^\infty)$. And let $0 \in B_p(\omega)$ represent the identity element $[0]$ of $\mathbb{Z}(p^\infty)$. Define $\mathcal{G}(p^\infty)$ to be the group with universe $B_p(\omega)$ (whose elements are written in reverse p-ary, as usual) and whose identity element is 0. And given elements $x_0x_1\cdots x_s$ and $y_0y_1\cdots y_t$ of $\mathcal{G}(p^\infty)$, the addition $\oplus$ in $\mathcal{G}(p^\infty)$ is defined as follows: Assuming without loss of generality that $s > t$, we have $x_0x_1\cdots x_s \oplus y_0y_1\cdots y_t = (x_0x_1\cdots x_{t-1}x_t + y_0y_1\cdots y_{t-1}y_t) * x_{t+1} * x_{t+2}\cdots * x_s$, where $+$ is the ordinary p-ary addition with carrying involved from the tth symbol to the $(t-1)$th symbol, and so on all the way back to the first symbol $x_0 + y_0$, which is then written modulo p. Furthermore, no terminal segment of 0's are included. As an example, suppose $p = 5$ and let the three strings 4020111, 3443, and 0002 be elements of $\mathcal{G}(p^\infty)$. Then $4020111 \oplus 3443 = 3013111$, and $3443 \oplus 0002 = 4$.

We now claim that $\mathbb{Z}(p^\infty)$ is represented by $\mathcal{G}(p^\infty)$. To see this, we first note that for each $n \in \mathbb{N}$, the distinct elements generated by $[1/p^n]$, namely, the $p^n - 1$ elements $[x/p^n]$, where $1 \leqslant x \leqslant p^n - 1$, correspond to elements of $\mathcal{G}(p^\infty)$ according to the following inductive rule: $[1/p^n]$ corresponds to the string $0^{n-1}1$, while $[2/p^n]$ is represented by the string $0^{n-1}2$, and, in general, if $[(x-1)/p^n]$ is represented by the string σ, then $[x/p^n]$ is represented by the string $\sigma \oplus 1$. So, for example, $[(p-1)/p^n]$ is represented by $(0^{n-1})^\frown(p-1)$, and $[(p+1)/p^n] = [p/p^n] + [1/p^n]$ is represented by $0^{n-2}1 \oplus 0^{n-1}1 = 0^{n-2}11$, while $[(p^n-1)/p^n]$ is represented by $(p-1)^\frown(p-1)^\frown\cdots^\frown(p-1)$, where $p-1$ occurs n times. Now to check that $\mathbb{Z}(p^\infty)$ is completely represented by $\mathcal{G}(p^\infty)$, it suffices to check that addition is preserved among the generators of either group by the above correspondence. Consider two generators $[1/p^n]$ and $[1/p^m]$ of $\mathbb{Z}(p^\infty)$, and assume without loss of generality that $m > n$. The corresponding "generators" in $\mathcal{G}(p^\infty)$ are $0^{n-1}1$ and $0^{m-1}1$. In $\mathbb{Z}(p^\infty)$, we have $[1/p^n] + [1/p^m] = [1/p^n + 1/p^m]$, while in $\mathcal{G}(p^\infty)$, we have $0^{n-1}1 \oplus 0^{m-1}1 = 0^{n-1}10^{m-n-1}1$. Since the representation of $[1/p^n + 1/p^m]$ in $\mathcal{G}(p^\infty)$ is $0^{n-1}10^{m-n-1}1$, we see that addition is preserved among generators by the above correspondence.

To finish the proof for $\mathbb{Z}(p^\infty)$, we observe that the group $\mathcal{G}(p^\infty) = (B_p(\omega), \oplus, 0)$, is a LOGSPACE group because its addition $\oplus$ is, more-or-less, p-ary addition, which, moreover, has the property that $|a \oplus b| \leqslant \max(|a|, |b|)$. Now by Lemma 2.3.10 and Lemma 3.1.1, the group $\mathcal{G}(p^\infty)$ is recursively isomorphic to a LOGSPACE group $\mathcal{H}$ with universe $\mathrm{Bin}(\omega)$ and whose addition is such that $|a +^H b| \leqslant c(|a| +^H |b|)$, for some constant $c > 0$. Hence by Lemma 3.1.3, the group $\mathcal{H}$ is also recursively isomorphic to a group with universe $\mathrm{Tal}(\omega)$.

Finally, we prove the result for the group $\mathbb{Q}$. Each rational number r is the sum of an integer $\lfloor r \rfloor$ and a *positive* proper fraction of the form $a/(p_1^{m_1} p_2^{m_2} \cdots p_n^{m_n})$, where the p_i are primes and the $m_i \geqslant 1$, and where a is an integer strictly less than $p_1^{m_1} p_2^{m_2} \cdots p_n^{m_n}$. Now let p and q be prime numbers and let $m, n \geqslant 1$. By using the Euclidean algorithm in reverse and back substitution, we can effectively write the greatest common divisor (i.e., 1) of p^m and q^n as the linear combination $1 = xp^m + yq^n$, where x and y are (possibly negative) integers. Thus if $|y| < p^m$ and $|x| < q^n$, we can effectively represent $1/(p^m q^n) = (y/p^m) + (x/q^n)$ as the element $\langle [y/p^m], [x/q^n] \rangle$ of the group $\mathbb{Z}(p^\infty) \oplus \mathbb{Z}(q^\infty)$. If $|y| \geqslant p^m$, that is, $y = bp^m + c$, and $|x| < q^n$, then we can effectively represent $1/(p^m q^n) = (y/p^m) + (x/q^n)$ as the element $\langle b, [c/p^m], [x/q^n] \rangle$ of the group $\mathbb{Z} \oplus \mathbb{Z}(p^\infty) \oplus \mathbb{Z}(q^\infty)$. Similarly, we can effectively represent $1/(p^m q^n)$ as an element of the group $\mathbb{Z} \oplus \mathbb{Z}(p^\infty) \oplus \mathbb{Z}(q^\infty)$ if either $|x| \geqslant q^n$ only, or both $|x| \geqslant q^n$ and $|y| \geqslant p^m$. It follows that we can effectively represent $1/(p_1^{m_1} p_2^{m_2} \cdots p_n^{m_n})$ as an element of $\mathbb{Z} \oplus \mathbb{Z}(p_1^\infty) \oplus \mathbb{Z}(p_2^\infty) \oplus \cdots \oplus \mathbb{Z}(p_n^\infty)$. These considerations show that every rational can be uniquely and recursively represented as an element of the group $\mathbb{Z} \oplus \left(\bigoplus_{\text{primes } p} \mathbb{Z}(p^\infty)\right)$. The addition operation in $\mathbb{Q}$ can be represented in $\mathbb{Z} \oplus \left(\bigoplus_{\text{primes } p} \mathbb{Z}(p^\infty)\right)$ with coordinatewise addition and with carrying involved, *thus making it different from the usual addition operation of* $\mathbb{Z} \oplus \left(\bigoplus_{\text{primes } p} \mathbb{Z}(p^\infty)\right)$. Moreover, the group $\mathbb{Z} \oplus \left(\bigoplus_{\text{primes } p} \mathbb{Z}(p^\infty)\right)$ is recursively isomorphic to a LOGSPACE group with universe $\mathrm{Bin}(\omega)$ and to a LOGSPACE group with universe $\mathrm{Tal}(\omega)$. This is because the group $\mathbb{Z}$ and each of the groups $\mathbb{Z}(p^\infty)$ are fully uniformly LOGSPACE over both $\mathrm{Tal}(\omega)$ and $\mathrm{Bin}(\omega)$, and the hypotheses of parts (d) and (e) of Lemma 3.3.2 apply.

□

We refer to the groups $\mathbb{Z}$, $\bigoplus_\omega \mathbb{Z}_k$, $\mathbb{Z}(p^\infty)$, and $\mathbb{Q}$, together with the finite Abelian groups, as the *basic groups.* As an immediate consequence of the previous lemma, Lemma 2.3.14, and Lemma 3.1.1, we have the following result:

Lemma 3.3.4. *Any finite product of basic groups is recursively isomorphic to a LOGSPACE group with universe* $\mathrm{Tal}(\omega)$ *and to a LOGSPACE group with universe* $\mathrm{Bin}(\omega)$.

A classical theorem of Abelian groups says that any finitely generated Abelian group is isomorphic to a finite product of cyclic groups which are either $\mathbb{Z}$ or $\mathbb{Z}_k$. Moreover, by considering generators, it is easy to see that any two finitely generated recursive Abelian groups that are isomorphic to one another are, in fact, recursively isomorphic to one another. Hence, the previous lemma now implies the following result:

Theorem 3.3.5. *Any finitely generated recursive Abelian group is recursively isomorphic to a LOGSPACE Abelian group with universe* $\mathrm{Tal}(\omega)$ *and to a LOGSPACE Abelian group with universe* $\mathrm{Bin}(\omega)$.

A natural question arises at this point, namely, whether any group $\mathcal{G}$ which is isomorphic to a LOGSPACE group $\mathcal{H}$ is necessarily recursively isomorphic to $\mathcal{H}$. A recursive structure $\mathcal{A}$ is called *recursively categorical* (or *autostable* in the Russian school) if any recursive structure $\mathcal{B}$ which is isomorphic to $\mathcal{A}$ is in fact recursively isomorphic to $\mathcal{A}$. Smith [24] characterized the recursively categorical p-groups, that is, those groups in which every element has order a power of a fixed prime p. We now state Smith's result and use it to obtain the only (positive) categoricity result in our work.

Theorem 3.3.6 (Smith [24]). *A recursive p-group $\mathcal{G}$ is recursively categorical if and only if either*

(1) $\mathcal{G} \approx \bigoplus_\omega \mathbb{Z}(p^\infty) \oplus \mathcal{F}$ *or*

(2) $\mathcal{G} \approx \bigoplus_{i<n} \mathbb{Z}(p^\infty) \oplus \bigoplus_\omega \mathbb{Z}_{p^m} \oplus \mathcal{F}$, *where $\mathcal{F}$ is a finite p-group and $m, n \in \omega$.*

Corollary 3.3.7. *Any recursively categorical p-group is recursively isomorphic to a LOGSPACE group with universe* $\mathrm{Tal}(\omega)$ *and also to a LOGSPACE group with universe* $\mathrm{Bin}(\omega)$.

Proof: Since $\mathcal{F}$ is finite, evidently $\mathcal{F}$ is recursively isomorphic to a ZEROSPACE group with universe a finite subset of $\mathrm{Bin}(\omega)$ and to one with universe a finite subset of $\mathrm{Tal}(\omega)$. By Lemma 3.3.3, the group $\bigoplus_\omega \mathbb{Z}_{p^m}$ is recursively isomorphic to a LOGSPACE group with universe $\mathrm{Tal}(\omega)$ and to a LOGSPACE group with universe $\mathrm{Bin}(\omega)$. The same is true for $\bigoplus_{i<n} \mathbb{Z}(p^\infty)$ by Lemma 3.3.4. To finish the proof, we simply need to observe that the products $\bigoplus_\omega \mathbb{Z}(p^\infty) \oplus \mathcal{F}$ and $\bigoplus_{i<n} \mathbb{Z}(p^\infty) \oplus \bigoplus_\omega \mathbb{Z}_{p^m} \oplus \mathcal{F}$ are finite products of groups that are recursively isomorphic to LOGSPACE groups whose universes may be taken to be either $\mathrm{Bin}(\omega)$ or $\mathrm{Tal}(\omega)$, and so Lemma 2.3.14 and Lemma 3.1.1 apply.

□

The p-groups and the *p-primary* groups (i.e., Abelian groups whose orders are a power of a fixed prime p) are examples of *torsion* groups, that is, groups in which every element has finite order. A group is said to be *torsion-free* if all its nonidentity elements have infinite order. Note that torsion groups by definition must necessarily be Abelian. Another class of torsion groups are the finite cyclic groups. We do not have general results like "Every product of cyclic groups is recursively categorical" since the group $\mathcal{G} = \bigoplus_\omega \mathbb{Z}_2 \oplus \bigoplus_\omega \mathbb{Z}_4$ is not recursively categorical, as follows from Smith's [24] Theorem. Thus there is a recursive group $\mathcal{H}$ which is isomorphic to $\mathcal{G}$

but not recursively isomorphic to $\mathcal{G}$. The next theorem shows that $\mathcal{H}$ is nonetheless recursively isomorphic to a LOGSPACE group.

Theorem 3.3.8. *Any recursive torsion Abelian group* $\mathcal{G} = (G,\ +^G,\ -^G,\ e^G)$ *is recursively isomorphic to a LOGSPACE group with universe a subset of* $\mathrm{Bin}(\omega)$ *and to a LOGSPACE group with universe a subset of* $\mathrm{Tal}(\omega)$.

Proof: We may assume that the group $\mathcal{G}$ is infinite, that its universe is $G = \mathrm{Bin}(\omega)$, and that $e^G = 0$.

First, we will introduce a recursive ordering on G of order type ω. For each $k \in \omega$, let $G_k = \langle\{0, 1, \ldots, k\}\rangle$ be the subgroup of G generated by the elements $\mathrm{bin}(0)$, $\mathrm{bin}(1)$, $\ldots$, $\mathrm{bin}(k)$. Now for each $g \in G$, let $k(g)$ be the least k such that $g \in G_k$. We then order the elements of G as follows: We say that a precedes b if either (i) $k(a) < k(b)$, or (ii) $k(a) = k(b)$ and $|a| < |b|$, or (iii) $k(a) = k(b)$, $|a| = |b|$, and a is lexicographically less than b. Let $\{a_0, a_1, \ldots\}$ be the listing of elements of G in this ordering. Since all the elements of G have finite order and G is Abelian, this ordering is both recursive and has order type ω.

We take a detour to note that the hypothesis that G is Abelian is essential in the previous paragraph. Otherwise, the sum $1 +^G 2 +^G 1 +^G 2 +^G 1 +^G 2 +^G \cdots$, for example, may not terminate. More specifically, suppose G is the non-Abelian symmetric group on ω. Let $a, b \in G$, where for each $n \geqslant 0$, we have $a(2n) = 2n + 1$ and $a(2n + 1) = 2n$; and $b(0) = 0$, $b(2n + 1) = 2n + 2$, and $b(2n + 2) = 2n + 1$. Then $a +^G a = id$ and $b +^G b = id$, which implies $|a| = 2 = |b|$. However, $a +^G b +^G a +^G b +^G a +^G b +^G \cdots$ does not terminate. So the subgroup $\langle\{a, b\}\rangle$ of G generated by a and b is infinite, and hence the ordering of the previous paragraph is not recursive in this case. We now return to our proof.

Consider the group $\mathcal{A} = (A,\ +^A,\ -^A,\ 0)$, where $A = \mathrm{Tal}(\omega)$ and, for m, n, $p \in \omega$, we have $m\ \ +^A / -^A\ \ n = p$ if and only if $a_m\ \ +^G / -^G\ \ a_n = a_p$. Evidently $\mathcal{A}$ is isomorphic to $\mathcal{G}$ via the isomorphism $i \mapsto a_i$. Now for each k, let $\mathcal{A}_k$ be the subgroup of $(A, +^A, -^A, 0)$ generated by the set $\{0, 1, \ldots, k\}$. Then the universe A_k of $\mathcal{A}_k$ is an initial segment of A. This is because if $x \in A_k$, then $a_x \in \langle\{a_0, a_1, \ldots, a_k\}\rangle$ and hence, $x \in G_k$, which means for each $y < x$, we must also have $y \in G_k$, which in turn implies $a_y \in \langle\{a_0, a_1, \ldots, a_k\}\rangle$, and hence $y \in A_k$. Now suppose $i \ +^A / -^A \ j = k$ and $i \leqslant j$. Then $k \in A_j$, and since A_j is an initial segment of A, it follows that $\{0, 1, \ldots, k\} \subset A_j$, and hence $A_k \subset A_j$.

We will now proceed to define the LOGSPACE group $\mathcal{B} = (B, +^B, -^B, e^B)$ which is recursively isomorphic to $\mathcal{A}$ and therefore recursively isomorphic to $\mathcal{G}$. For each k, let $\nu(k)$ be the number of steps needed to write down each of 0, 1, …, k, and then compute and store each of the sums and differences $a \ +^A / -^A \ b$, where a and b range over A_k. Thus $\nu(k)$ is the total time needed to compute the operation table for A_k. Now let $\phi : A \to \text{Bin}(\omega)$ be defined by $\phi(k) = 0^{2^k} 1^{2^{\nu(k)}}$ and let $B = \{\phi(k) \in \text{Bin}(\omega) : k \in \omega\}$. Let $e^B = \phi(0)$ and define the operations $+^B$ and $-^B$ so as to make ϕ a group isomorphism.

Now we will show that B is a LOGSPACE set. Given a binary number on the input tape, our machine first checks that the number is of the form $0^{2^k} 1^{2^t}$, and then writes down k and t, using procedures described in the proof of Theorem 3.2.1. All this uses up at most logarithmic space. Then it begins to compute the operation table for $\langle\{0, 1, \ldots, k\}\rangle$ while keeping track of the number of steps being completed. The number on the input tape is in B if and only if the operation table is computed in exactly t steps. Since t is logarithmic in the length of the input, B is a LOGSPACE set.

Finally, given $0^{2^i} 1^{2^{\nu(i)}}$ and $0^{2^j} 1^{2^{\nu(j)}}$ on two input tapes, our machine computes $0^{2^k} 1^{2^{\nu(k)}} = 0^{2^i} 1^{2^{\ nu(i)}} +^B / -^B 0^{2^j} 1^{2^{\nu(j)}}$ as follows. The machine checks whether $i \leqslant j$ and then computes $k = i +^A / -^A j$, which takes time at most $\nu(j)$. (Note that of course $k \leqslant \nu(j)$). After that, the machine writes down 0, 1, … , k, and computes the operation table for A_k, keeping count of the number $\nu(k)$ of steps required. Since

$A_k \subset A_j$, we have $\nu(k) \leqslant \nu(j)$. Outputing $2^{\nu(k)}$ in tally requires space linear in $\nu(k)$. Thus the operations $+^B/-^B$ can be carried out in LOGSPACE.

To finish the proof, it suffices to note that the "tally version" of $\mathcal{B}$ is a LOGSPACE group by Lemma 3.1.3 since $\left|0^{2^i}1^{2^{\nu(i)}} +^B / -^B 0^{2^j}1^{2^{\nu(j)}}\right| \leqslant \left|0^{2^{\nu(j)}}1^{2^{\nu(j)}}\right| = 2^{\nu(j)+1} \leqslant 2\left|0^{2^j}1^{2^{\nu(j)}}\right|$.

□

The next question to consider is which torsion groups are recursively isomorphic to a LOGSPACE group with a specified universe such as Tal(ω) or Bin(ω). Our goal is to show that every recursive torsion Abelian group G with an upper bound on the orders of the elements of G is recursively isomorphic to a LOGSPACE Abelian group with universe either Tal(ω) or Bin(ω). But before reaching this goal, we need to prove a series of lemmas. We first reduce our problem to the case of p-groups. The Fundamental Theorem of Abelian groups implies the fact that every torsion Abelian group is a direct product of *finite* cyclic groups, which themselves can be decomposed to finite cyclic groups of prime power order. Hence, we have the following fact: *Every torsion Abelian group is a direct product of p-groups.* The next lemma is the LOGSPACE version of this fact.

Lemma 3.3.9. *Any recursive torsion Abelian group $\mathcal{G}$ is recursively isomorphic to a LOGSPACE direct product of p-groups that are fully uniformly LOGSPACE over B, where B may be taken to be either* Tal(ω) *or* Bin(ω).

Proof: We may assume that $\mathcal{G}$ is the LOGSPACE group with universe contained in Tal(ω), as constructed in the proof of the previous theorem. This means that, given $g \in G$, we can compute the tally representation tal$(o(g)) = 1^{o(g)}$ of the order $o(g)$ of g within space logarithmic in $|g|$. For each prime p, let G_p be the set of elements of G such that $o(g)$ is a power of p. Since $\mathcal{G}$ is torsion Abelian, it is well known from Group Theory that each $\mathcal{G}_p$ is a subgroup of $\mathcal{G}$ and that $\mathcal{G}$ is the direct product of the groups $\mathcal{G}_p$. Each of the groups $\mathcal{G}_p$ is a LOGSPACE group. Certainly $+^{G_p}/-^{G_p} = +^G/-^G$ are in LOGSPACE. And the universe G_p is in LOGSPACE because given g on the input tape, we can compose the LOGSPACE computation of $1^{o(g)}$ and the LOGSPACE division of $1^{o(g)}$ by $\text{tal}(p)$. Now let $\mathcal{A}_n = \mathcal{G}_{p_n}$, where p_n is the nth prime. Evidently $\mathcal{G}$ is recursively isomorphic to the direct product $\bigoplus_n \mathcal{A}_n$.

We now proceed to show that the sequence $\{\mathcal{A}_n\}$ of groups is fully uniformly LOGSPACE over B. Let $b(n) = \text{tal}(n)$ if $B = \text{Tal}(\omega)$ and $b(n) = \text{bin}(n)$ if $B = \text{Bin}(\omega)$.

To see that $R = \{\langle b(n), g\rangle : g \in A_n\}$ is a LOGSPACE subset of $B \times B$, we first note that since we can compute $1^{o(g)}$ within space logarithmic in $|g|$, we can certainly compute $1^{o(g)}$ within space logarithmic in $|\langle b(n), g\rangle|$. Now for any $m \in \omega$, we can test that m is a prime within space logarithmic in $m = 1^m$. Hence we can find the number of primes $\leqslant o(g)$ within space logarithmic in $|\langle b(n), g\rangle|$. Suppose there are k primes $\leqslant o(g)$. Then we can compute $b(k)$ and compare it to $b(n)$. This will once again use up at most logarithmic space. If $b(n) > b(k)$, then $\langle b(n), g\rangle \notin R$. If $b(n) \leqslant b(k)$, then we can use binary counters to find the nth prime $\text{tal}(p_n)$ in tally within space logarithmic in $|\langle b(n), g\rangle|$. Then by dividing $\text{tal}(o(g))$ by $\text{tal}(p_n)$, we can check within logarithmic space whether $o(g)$ is a multiple of p_n. If so, then $\langle b(n), g\rangle \in R$; otherwise $\langle b(n), g\rangle \notin R$. Thus R is a LOGSPACE set.

Now let $\mathcal{A}_n = (A_n, +_n, -_n, e_n)$. Since $+_n = +_G$, $-_n = -_G$, and G and $R = \{\langle b(n), g\rangle : g \in A_n\}$ are LOGSPACE sets, it follows that the functions $F(b(n), a, b) = a +_n b$ and $G(b(n), a, b) = a -_n b$ are both the restrictions of LOGSPACE functions from B^3 to B, where we set $F(b(n), a, b) = G(b(n), a, b) = \emptyset$ if either a or b is not in A_n. As for the function from $\text{Tal}(\omega)$ into B defined by $e(\text{tal}(n)) = e_n$, it is in ZEROSPACE since $e_n = e_G$ for all n. The lemma now follows from Lemma 3.3.2 (a).

$\square$

The above lemma does not automatically allow us to prove that a torsion Abelian group $\mathcal{G}$ is recursively isomorphic to a LOGSPACE group with a specified universe. This is because we do not have enough information about the p-groups $\mathcal{G}_p$ in the proof of the previous lemma. If, for example, we knew that $\mathcal{G}_p$ is nontrivial for at most finitely many p and that there is a p such that $\mathcal{G}_p$ is recursively isomorphic to a LOGSPACE group over (say) $\mathrm{Tal}(\omega)$, then we could use Lemma 3.3.2 (c) to to conclude that $\mathcal{G}$ is recursively isomorphic to a LOGSPACE group with universe $\mathrm{Bin}(\omega)$.

Returning to our goal for recursive Abelian torsion groups with an upper bound on the orders of the elements, we need some definitions. For an integer i and an element g of a group $\mathcal{G} = (G, +^G)$, we define $i \cdot g$ inductively as follows: Let $0 \cdot g = 0^G$, and for each $i > 0$, let $(i+1) \cdot g = i \cdot g +^G g$. Also, for $i > 0$, let $(-i) \cdot g = 0^G -^G i \cdot g$. Now elements $g_1, \ldots, g_n$ of $\mathcal{G}$ are said to be *dependent* if there exist integers $i_1, \ldots, i_n$ such that $i_1 \cdot g_1 +^G \cdots +^G i_n \cdot g_n = 0^G$ with not all $i_k \cdot g_k = 0^G$, and are said to be *independent* otherwise. An infinite set A of elements of $\mathcal{G}$ is said to be dependent if some finite subset of A is dependent; otherwise A is independent. Two subgroups $\mathcal{K}$ and $\mathcal{H}$ of $\mathcal{G}$ are *independent of each other* if $\mathcal{K} \cap \mathcal{H} = \{0^G\}$. And $\mathcal{K}$ is said to be *maximal independent* of $\mathcal{H}$ if $\mathcal{K}$ is independent of $\mathcal{H}$ and for any subgroup $\mathcal{J}$ of $\mathcal{G}$ such that $\mathcal{H}$ is a proper subgroup of $\mathcal{J}$, the subgroup $\mathcal{J}$ is not independent of $\mathcal{H}$.

We now quote Lemmas 4.9, 4.10, and 4.11 that are stated and proved in Cenzer and Remmel [3].

Lemma 3.3.10. *Let $\mathcal{G}$ be a recursive p-group of bounded order which is isomorphic to a direct product $\bigoplus_i (\mathbb{Z}_{p^i})^{e(i)}$ with each $e(i) \leqslant \omega$, and let p^m be the maximal order of an element of G. If $e(m) = \omega$, then there is a recursive subgroup $\mathcal{H}$ of $\mathcal{G}$ which is generated by an infinite independent recursive set of elements all of order p^m.*

Lemma 3.3.11. *Let $\mathcal{G}$ be a recursive Abelian torsion group and let $\mathcal{H}$ be a recursive subgroup of $\mathcal{G}$. Then there is a recursive subgroup $\mathcal{K}$ of $\mathcal{G}$ which is maximal independent of $\mathcal{H}$.*

Lemma 3.3.12. *Let $\mathcal{G}$ be an Abelian group such that for some fixed prime p and natural number m we have $p^m \cdot x = 0^G$ for all $x \in G$. Let $\mathcal{H}$ be a subgroup of $\mathcal{G}$ generated by some independent set of elements all of order p^m, and let $\mathcal{K}$ be a subgroup of $\mathcal{G}$ which is maximally independent of $\mathcal{H}$. Then $\mathcal{G} = \mathcal{H} \oplus \mathcal{K}$.*

We are now ready to prove our main theorem on recursive Abelian groups whose elements have bounded order.

Theorem 3.3.13. *Let $\mathcal{G}$ be an infinite recursive Abelian group with an upper bound on the orders of its elements. Then $\mathcal{G}$ is recursively isomorphic to a LOGSPACE group with universe* Tal(ω) *and to a LOGSPACE group with universe* Bin(ω).

Proof: Fix $B = \text{Tal}(\omega)$ or $\text{Bin}(\omega)$. Since $\mathcal{G}$ is torsion and the orders of the elements of $\mathcal{G}$ are bounded, there is a finite set P of primes such that $\mathcal{G}$ is recursively isomorphic to the finite direct sum $\bigoplus_{p\in P} \mathcal{G}_p$. At least one of the primary components $\mathcal{G}_p$ of the finite sum $\bigoplus_{p\in P} \mathcal{G}_p$ must be infinite. By Lemma 3.3.2, it suffices to show that the result is true for this particular p-group $\mathcal{G}_p$. For the remainder of the proof, we simply denote $\mathcal{G}_p$ by $\mathcal{G}$. Now there is natural number m such that $p^m \cdot x = 0^G$ for all $x \in G = G_p$. The result is proved by induction on m. There are two possible cases.

Case 1. Suppose that $e(m)$ is finite. Then the subgroup $\mathcal{H}$ generated by the elements of order p^m is finite. By Lemma 3.3.11, there is a recursive subgroup $\mathcal{K}$ of $\mathcal{G}$ that is maximal independent of $\mathcal{H}$. And by Lemma 3.3.12, we have $\mathcal{G} = \mathcal{H} \oplus \mathcal{K}$. Since the elements of $\mathcal{K}$ have order bounded by p^{m-1}, the induction hypothesis implies $\mathcal{K}$ is recursively isomorphic to a LOGSPACE group with universe B. Since $\mathcal{H}$ is finite and hence a ZEROSPACE group, it follows from Lemma 2.3.14 and Lemma 3.1.1 that $\mathcal{G} = \mathcal{H} \oplus \mathcal{K}$ is recursively isomorphic to a LOGSPACE group with universe B.

Case 2. Now suppose $e(m)$ is infinite. By Lemma 3.3.10, there is a recursive subgroup $\mathcal{H}$ of $\mathcal{G}$ that is generated by an infinite and independent recursive set of elements all of order p^m. Hence $\mathcal{H}$ is isomorphic to $\bigoplus_\omega \mathbb{Z}(p^m)$. By Lemma 3.3.11, there is a recursive subgroup $\mathcal{K}$ of $\mathcal{G}$ which is maximal independent of $\mathcal{H}$. And again by Lemma 3.3.12, we have $\mathcal{G} = \mathcal{H} \oplus \mathcal{K}$. By Theorem 3.3.8, we may assume that $\mathcal{K}$ is in LOGSPACE with universe a subset of B, and by Corollary 3.3.7, we may assume that $\mathcal{H}$ is in LOGSPACE with universe B. Hence by Lemma 2.3.14 and Lemma 3.1.1 we can conclude once again that $\mathcal{G} = \mathcal{H} \oplus \mathcal{K}$ is recursively isomorphic to a LOGSPACE group with universe B.

□

For torsion Abelian groups with no upper bound on the orders of the elements, there happen to be examples of recursive groups that are recursively isomorphic to LOGSPACE groups with standard universe, and also examples of recursive groups that are in fact not recursively isomorphic to any LOGSPACE group. Our investigation here parallels that for permutation structures in that we have positive results for groups with certain spectra. So we now define the notions of spectrum and monic for groups.

Given a recursive Abelian group $\mathcal{G}$ with universe G, we define the *spectrum* of $\mathcal{G}$ by Spec($\mathcal{G}$) = {prime powers $p^n \in \omega : (\exists a \in G)\ |a|_G = p^n$}. Evidently Spec($\mathcal{G}$) is an r.e. subset of ω. It is well known from Group Theory that for any prime p and any j, if $p^j \in$ Spec($\mathcal{G}$)and $i < j$, then $p^i \in$ Spec($\mathcal{G}$). A set P of prime powers is said to be *hereditary* if for all primes p and natural numbers j, we have $(p^j \in P \wedge i < j) \longrightarrow p^i \in P$. And P is said to be *locally bounded* if for all primes p, the set $\{i : p^i \in P\}$ is bounded. We define the *finite and infinite parts* of $\mathcal{G}$ by Fin($\mathcal{G}$) = $\{a \in G : |a|_g < \omega\}$ and Inf($\mathcal{G}$) = $\{a \in G : |a|_g = \omega\}$. It is evident that Fin(G) is an r.e. subset of G and that each finitely generated subgroup of G is an r.e. subset of G. A group $\mathcal{G}$ is said to be *monic* if $\mathcal{G}$ is a torsion Abelian group such that for any prime p, the subgroup $\mathcal{G}_p$ is cyclic. Recall that $\mathcal{G}_p$ is the subgroup all of whose elements have order a power of p. Thus for any monic group $\mathcal{G}$ and any number n, there is at most one subgroup of $\mathcal{G}$ of order n.

We now quote Theorem 4.13 in Cenzer and Remmel [3] which shows that in order to examine the possible spectra of a recursive group, it is reasonably general to restrict ourselves primarily to monic groups.

Theorem 3.3.14. (a) *For any nonempty r.e. hereditary set P of prime powers, there is a recursive Abelian group $\mathcal{G}$ with* $\mathrm{Spec}(\mathcal{G}) = P$. *Furthermore, if P is locally bounded, then $\mathcal{G}$ is monic.*
(b) *Let $\mathcal{A}$ and $\mathcal{B}$ be recursive monic torsion Abelian groups having the same recursive spectrum P. If P is locally bounded, then $\mathcal{A}$ and $\mathcal{B}$ are recursively isomorphic.*

Theorem 3.3.15. *Let* $B = \mathrm{Tal}(\omega)$ *or* $\mathrm{Bin}(\omega)$. *For any nonempty infinite hereditary locally bounded set Q of prime powers such that* $\mathrm{tal}(Q)$ *is in LOGSPACE, there is a monic LOGSPACE Abelian torsion group $\mathcal{B}$ with universe B and with* $\mathrm{Spec}(\mathcal{B}) = Q$.

Proof: Part I. We first do the proof for $B = \mathrm{Tal}(\omega)$. Define the subset P of Q by $P = \{p^i \in Q : p \text{ is prime and } p^{i+1} \notin Q\}$. We claim that $\mathrm{tal}(P)$ is a LOGSPACE subset of $\mathrm{Tal}(\omega)$. To test if $x \in \mathrm{tal}(P)$, we first test if $x \in \mathrm{tal}(Q)$, which uses up only logarithmic space. If $x \notin \mathrm{tal}(Q)$, then certainly $x \notin \mathrm{tal}(P)$. But if $x \in \mathrm{tal}(Q)$, then $x = \mathrm{tal}(p^i)$ for some prime p and some i. The idea is to compute p and i, and then to compute $\mathrm{tal}(p) \cdot x = \mathrm{tal}(p^{i+1})$ and to determine if $\mathrm{tal}(p^{i+1}) \in \mathrm{tal}(Q)$. Hence it suffices to show that p, i, and $\mathrm{tal}(p) \cdot x = \mathrm{tal}(p^{i+1})$ can all be computed in LOGSPACE. Certainly $\mathrm{tal}(p) \cdot x = \mathrm{tal}(p^{i+1})$ can be computed in ZEROSPACE. As for p and i, we first test whether $\mathrm{tal}(x)$ is prime. To do this, we write each number $n < x$ in binary on three work tapes, and then advance the input cursor once to the right each time we subtract 1 in binary from $\mathrm{bin}(n)$, and keep doing this until we encounter a $\sqcup$ or the last 1 of x. Evidently we can find a divisor d of x or determine whether x is prime using this LOGSPACE procedure. If x is prime, then $p = x$ and $i = 1$. Otherwise, we have $p = d$ and $\mathrm{bin}(d)$ is written on a work tape. We compose the LINSPACE conversion of $\mathrm{bin}(d)$ to $\mathrm{tal}(d)$ and the ZEROSPACE division of $\mathrm{tal}(x)$ by $\mathrm{tal}(d)$ to yield i. This composition is linear in $|\text{bin}(d)|$, and hence logarithmic in the length of the input $\text{tal}(x)$. Thus we have proved our claim.

Now let the set P be enumerated in increasing order as $p_0 < p_1 < \cdots$, where each p_i is a prime power. A natural monic Abelian group with spectrum Q is the direct sum $\mathcal{G}$ of the sequence $\mathcal{G}_n = \mathbb{Z}_{p_n}$. This is because for each prime power $p^k \in Q$, we have a power $p^i \in P$ with $i \geqslant k$ and $\mathcal{G}$ will have a component group $\mathbb{Z}_{p^i}$, and hence $\mathcal{G}$ will have an element of order p^k as well. We represent the group $\mathbb{Z}_{p_n}$ as the set of integers $\{\text{tal}(0), \text{tal}(1), \ldots, \text{tal}(p_n - 1)\}$ with the operation being addition modulo p_n, which is a ZEROSPACE operation. Thus the universe G of $\mathcal{G}$ will consist of the empty string $\emptyset$ which represents the 0 of $\mathcal{G}$, and all codes of finite sequences $\langle \text{tal}(i_0), \ldots, \text{tal}(i_k)\rangle$ such that $i_t < p_t$ for each $t \leqslant k$ and $i_k \neq 0$. The group operation of $\mathcal{G}$ is defined componentwise in the obvious fashion. Note that $\mathcal{G}$ is not a LOGSPACE structure since the subtraction operation is not in LOGSPACE. For example, the inverse of the element $\langle 0, 0, \ldots, 0, 1\rangle$ (with $k+1$ components) is $\langle 0, 0, \ldots, 0, p_k - 1\rangle$. We cannot, in general, compute p_k within logarithmic space even though the set P is in LOGSPACE because p_k may be much larger than the length of $\langle 0, 0, \ldots, 0, 1\rangle$.

Now we proceed to define a LOGSPACE Abelian group $\mathcal{B}$ with universe $\text{Tal}(\omega)$, and show that $\mathcal{B}$ is isomorphic to $\mathcal{G}$ and, therefore, is a monic torsion group with spectrum Q.

It is easy to show that each $n > 0$ has a unique representation in the form $n = i_0 + i_1 \cdot p_0 + \cdots + i_k \cdot p_0 \cdot p_1 \cdots p_{k-1}$, where $i_t < p_t$ for $t \leqslant k$ and $i_k > 0$. Hence the most natural map from G to B would be the one that sends the element $\langle \text{tal}(i_0), \ldots, \text{tal}(i_k)\rangle$ of G to the element $\text{tal}(i_0 + i_1 \cdot p_0 + \cdots + i_k \cdot p_0 \cdot p_1 \cdots p_{k-1})$ of B. Unfortunately, if the operations in $\mathcal{B}$ are defined so as to make this map a group-isomorphism, then the operations are not necessarily polynomial-time and hence not necessarily in LOGSPACE. For example, we must have $0^B -^B p_0 \cdot p_1 \cdots p_{k-1} = (p_k - 1) \cdot p_0 \cdot p_1 \cdots p_{k-1}$, since $0^B -^B p_0 \cdot p_1 \cdots p_{k-1}$ represents the inverse of the element $\langle 0, 0, \ldots, 0, 1\rangle$ (with $k+1$ components) of $\mathcal{G}$ considered above. But p_k may be much larger than $p_0 \cdot p_1 \cdots p_{k-1}$ for infinitely many k so that it is not possible to carry out this computation in polynomial time. Hence instead of using the above natural map, we will modify the map in such a way that whenever the computation of a sum or difference in G requires us to replace a kth coordinate entry x by $p_k - x$, the number p_k will be less than the sum of the two kth coordinates involved in the operation.

More precisely, we define the map $\phi : \mathcal{G} \to \mathcal{B}$ as follows: First $\phi(\emptyset) = 0$. Next if $\langle \mathrm{tal}(i_0), \ldots, \mathrm{tal}(i_k)\rangle \in G$ so that $i_t < p_t$ for $t \leqslant k$ and $i_k > 0$, then $\phi(\langle \mathrm{tal}(i_0), \ldots, \mathrm{tal}(i_k)\rangle) = \mathrm{tal}(j_0 + j_1 \cdot p_0 + \cdots + j_k \cdot p_0 \cdot p_1 \cdots p_{k-1})$, where for each $t \leqslant k$ we have $j_t = 2i_t$ if $2i_t < p_t$, and $j_t = 2(p_t - i_t) - 1$ if $2i_t \geqslant p_t$. It is easy to check that $j_t < p_t$ for all $t \leqslant k$ and that ϕ is a bijection. And the group operations on B are defined so as to make ϕ a group-isomorphism.

Note that it is not enough to prove that both ϕ and ϕ^{-1} can be computed within logarithmic space, and then invoke Lemma 3.1.1. This is because $\mathcal{G}$ is not a LOGSPACE structure. Hence, we must examine in detail the group operations induced on $\mathcal{B}$ and prove that these operations are in LOGSPACE.

Let $x = j_0 + j_1 p_0 + \cdots + j_k p_0 \cdots p_{k-1}$ and $y = b_0 + b_1 p_0 + \cdots + b_l p_0 \cdots p_{l-1}$, where $k \leqslant l$. Suppose $\mathrm{tal}(x) +^B \mathrm{tal}(y) = \mathrm{tal}(c_0 + c_1 p_0 + \cdots + c_t p_0 \cdots p_{t-1})$ and $\mathrm{tal}(x) -^B \mathrm{tal}(y) = \mathrm{tal}(d_0 + d_1 p_0 + \cdots + d_s p_0 \cdots p_{s-1})$. How should we define $c_0, \ldots, c_t$ and $d_0, \ldots, d_s$ so as to remain consistent with the group operations induced on $\mathcal{B}$ by ϕ? We must of course answer this question in such a way that the $c_0, \ldots, c_t$ and $d_0, \ldots, d_s$ can be obtained from $x = j_0 + j_1 p_0 + \cdots + j_k p_0 \cdots p_{k-1}$ and $y = b_0 + b_1 p_0 + \cdots + b_l p_0 \cdots p_{l-1}$ within logarithmic space. To that end, we first describe how, given $x = j_0 + j_1 p_0 + \cdots + j_k p_0 \cdots p_{k-1}$ and $y = b_0 + b_1 p_0 + \cdots + b_l p_0 \cdots p_{l-1}$ in tally on two input tapes, we can compute the $j_0, \ldots, j_k$ and $b_0, \ldots, b_l$ within logarithmic space. We will call this algorithm "PROCEDURE" for convenience.

We begin PROCEDURE by writing down bin(x) on a work tape. Since tal(P) is in LOGSPACE, we can write down bin(0), bin(1), bin(2), ..., one after another, and compose the conversion of these to tally and the checking of whether the tally numbers are in P. Whenever tal(n) $\in P$ for some bin(n), we save bin(n) on a separate work tape T, separated from the previously saved binary number by a $\sqcup$. On another work tape, we multiply these binary numbers on tape T as soon as a new number is written on T to obtain, say, $\text{bin}(n_0) \cdot \text{bin}(n_1) \cdots \text{bin}(n_\alpha)$ and also $2 \cdot \text{bin}(n_0) \cdot \text{bin}(n_1) \cdots \text{bin}(n_\alpha)$. All this uses up logarithmic space. As soon as we arrive at the least α such that $\text{bin}(n_0) \cdot \text{bin}(n_1) \cdots \text{bin}(n_\alpha) < \text{bin}(x)$ but $2 \cdot \text{bin}(n_0) \cdot \text{bin}(n_1) \cdots \text{bin}(n_\alpha) \geqslant \text{bin}(x)$, we know that we have in fact obtained the elements $\text{bin}(p_0), \ldots, \text{bin}(p_{k-1})$ such that $x = j_0 + j_1 p_0 + \cdots + j_k p_0 \cdots p_{k-1}$. Note that we have explicitly written down the numbers $\text{bin}(p_0), \ldots, \text{bin}(p_{k-1})$ and the product $\text{bin}(p_0) \cdot \text{bin}(p_1) \cdots \text{bin}(p_{k-1})$. Now we compose the conversion of $\text{bin}(p_0) \cdot \text{bin}(p_1) \cdots \text{bin}(p_{k-1})$ to tally and the division of tal(x) by $\text{tal}(p_0 \cdots p_{k-1})$, followed by the conversion and the writing down of the quotient (which is $\text{bin}(j_k)$) and the remainder r_1 in binary. Next, we carry out PROCEDURE on $\text{bin}(r_1)$, at the end of which we will have written down $\text{bin}(j_{k-1})$ and another remainder $\text{bin}(r_2)$. We keep carrying out PROCEDURE on the remainders $\text{bin}(r_i)$ until we end up with a zero remainder, at which point we will have written down $\text{bin}(j_0)$. Similarly, we explicitly write down the $\text{bin}(b_0), \ldots, \text{bin}(b_l)$ corresponding to $y = b_0 + b_1 p_0 + \cdots + b_l p_0 \cdots p_{l-1}$.

All we need to remember from the previous paragraph is as follows: Given $x = j_0 + j_1 p_0 + \cdots + j_k p_0 \cdots p_{k-1}$ and $y = b_0 + b_1 p_0 + \cdots + b_l p_0 \cdots p_{l-1}$ in tally on two input tapes, we can, using the algorithm PROCEDURE, explicitly write down the corresponding numbers $\text{bin}(p_0), \ldots, \text{bin}(p_{k-1})$ and $\text{bin}(p_0), \ldots, \text{bin}(p_{l-1})$, and the numbers $\text{bin}(j_0), \ldots, \text{bin}(j_k)$ and $\text{bin}(b_0), \ldots, \text{bin}(b_l)$, all within logarithmic space.

We now return to the question of how the $c_0, \ldots, c_t$ and $d_0, \ldots, d_s$ are to be defined, where $\text{tal}(x) +^B \text{tal}(y) = \text{tal}(c_0 + c_1 p_0 + \cdots + c_t p_0 \cdots p_{t-1})$ and $\text{tal}(x) -^B \text{tal}(y) =$

$\mathrm{tal}(d_0 + d_1p_0 + \cdots + d_sp_0 \cdots p_{s-1})$. We first consider how to compute $0^B -^B \mathrm{tal}(y)$ so that afterwards we need only consider the computation $\mathrm{tal}(x) +^B \mathrm{tal}(y)$ given that $\mathrm{tal}(x) -^B \mathrm{tal}(y) = \mathrm{tal}(x) +^B [0^B -^B \mathrm{tal}(y)]$.

We recall that the element $\mathrm{tal}(y) \in B$ corresponds to $\langle \mathrm{tal}(b_0), \ldots, \mathrm{tal}(b_l)\rangle$, which in turn corresponds to the element $\langle \mathrm{tal}(a_0), \ldots, \mathrm{tal}(a_l)\rangle \in G$, where for each $r \leqslant l$, we have $b_r = 2a_r$ if $2a_r < p_r$, and $b_r = 2(p_r - a_r) - 1$ if $2a_r \geqslant p_r$, and $a_r \in \mathbb{Z}_{p_r}$. Hence $0^B -^B \mathrm{tal}(y)$ corresponds to $\langle \mathrm{tal}(-a_0), \ldots, \mathrm{tal}(-a_l)\rangle \in G$, which is in fact $\langle \mathrm{tal}(p_0 - a_0), \ldots, \mathrm{tal}(p_l - a_l)\rangle \in G$. So for each $0 \leqslant r \leqslant l$, we must have d_r corresponding to $p_r - a_r$. We have two cases: (i) If $2(p_r - a_r) < p_r$, i.e., $2a_r > p_r$, then we must define $d_r = 2(p_r - a_r)$. Either $2a_r < p_r$ (which is impossible in this case) or $2a_r \geqslant p_r$, which, in this case, forces $2a_r > p_r$, which means $b_r = 2(p_r - a_r) - 1$, and therefore $d_r = 2(p_r - a_r) = b_r + 1$. (ii) If $2(p_r - a_r) \geqslant p_r$ i.e., $2a_r \leqslant p_r$, then we must define $d_r = 2(p_r - (p_r - a_r)) - 1 = 2a_r - 1$. Once again, either we have $2a_r < p_r$ or $2a_r \geqslant p_r$, which, in this case leads to two possibilities: $2a_r < p_r$ or $2a_r = p_r$. If $2a_r < p_r$, then $b_r = 2a_r$ and so $d_r = b_r - 1$. If $2a_r = p_r$, then $b_r = 2(p_r - a_r) - 1 = 4a_r - 2a_r - 1 = d_r$. To summarize, we have:

$$d_r = \begin{cases} b_r - 1 & \text{if } b_r \text{is even,} \\ b_r & \text{if } b_r + 1 = p_r, \\ b_r + 1 & \text{if otherwise} \end{cases}$$

Now we will discuss how the d_r's and then $0^B -^B \mathrm{tal}(y)$ can be calculated within logarithmic space given $\mathrm{tal}(y)$ as input. Recall that by using PROCEDURE, we explicitly write down the corresponding numbers $\mathrm{bin}(p_0)$, $\ldots$, $\mathrm{bin}(p_{l-1})$ and $\mathrm{bin}(b_0), \ldots, \mathrm{bin}(b_l)$ within logarithmic space. For each $0 \leqslant r \leqslant l$, checking whether $\mathrm{bin}(b_r)$ is even requires no space. And to check whether $b_r + 1 = p_r$ for $r = l$, we simply add 1 to $\mathrm{bin}(b_l)$, and then compose the linear-space conversion of $\mathrm{bin}(b_l + 1)$ to $\mathrm{tal}(b_l + 1)$ and the logarithmic-space testing of whether $\mathrm{tal}(b_l + 1) \in P$. It now follows that we can explicitly write down each $\mathrm{bin}(d_r)$. Since $\mathrm{bin}(p_0)$, $\ldots$, $\mathrm{bin}(p_{l-1})$ are already written, computing and writing down $\mathrm{bin}(d_0 + d_1p_0 + \cdots + d_lp_0 \cdots p_{l-1})$

uses up logarithmic-space and conversion to $\mathrm{tal}(d_0 + d_1p_0 + \cdots + d_lp_0 \cdots p_{l-1})$ uses up space logarithmic in $|\mathrm{tal}(y)|$.

Finally, we consider the computation of $\mathrm{tal}(x) +^B \mathrm{tal}(y) = \mathrm{tal}(j_0 + j_1p_0 + \cdots + j_kp_0 \cdots p_{k-1}) + \mathrm{tal}(b_0 + b_1p_0 + \cdots + b_lp_0 \cdots p_{l-1}) = \mathrm{tal}(c_0 + c_1p_0 + \cdots + c_tp_0 \cdots p_{t-1})$, where $k \leqslant l$. We reiterate for convenience that the element $\mathrm{tal}(x) \in B$ corresponds to $\langle \mathrm{tal}(j_0), \ldots, \mathrm{tal}(j_k)\rangle$, which in turn corresponds to the element $\langle \mathrm{tal}(i_0), \ldots, \mathrm{tal}(i_k)\rangle \in G$, where for each $r \leqslant k$, we have $j_r = 2i_r$ if $2i_r < p_r$, and $j_r = 2(p_r - i_r) - 1$ if $2i_r \geqslant p_r$. Similarly, the element $\mathrm{tal}(y) \in B$ corresponds to $\langle \mathrm{tal}(b_0), \ldots, \mathrm{tal}(b_l)\rangle$, which in turn corresponds to the element $\langle \mathrm{tal}(a_0), \ldots, \mathrm{tal}(a_l)\rangle \in G$, where for each $r \leqslant l$, we have $b_r = 2a_r$ if $2a_r < p_r$, and $b_r = 2(p_r - a_r) - 1$ if $2a_r \geqslant p_r$. We have two cases and four subcases within the first case.

Case 1. Suppose $k < l$. Then evidently we have $t = l$ and $c_r = b_r$ for $k < r \leqslant l$.

Next, for $r \leqslant k$, we define c_r depending on whether j_r and b_r are even or odd. There are four subcases. We will give the details only for the first two subcases, those for the other two being similar.

Subcase 1. Suppose $j_r = 2i_r$ and $b_r = 2a_r$. Then by definition of j_r and b_r, we have $2i_r < p_r$ and $2a_r < p_r$. Now c_r must represent the element $i_r + a_r \in \mathbb{Z}_{p_r}$. So if $2(i_r + a_r) < p_r$, then we must define $c_r = 2(i_r + a_r) = j_r + b_r$. And if $2(i_r + a_r) \geqslant p_r$, then $c_r = 2(p_r - (i_r + a_r)) - 1 = 2p_r - (j_r + b_r) - 1$. To summarize, we have

$$c_r = \begin{cases} j_r + b_r & \text{if } \; j_r + b_r < p_r, \\ 2p_r - (j_r + b_r) - 1 & \text{if } \; j_r + b_r \geqslant p_r. \end{cases}$$

Subcase 2. Suppose $j_r = 2i_r$ and $b_r = 2(p_r - a_r) - 1$. Then by definition of j_r and b_r, we have $2i_r < p_r$ and $2a_r \geqslant p_r$. Since c_r must represent $i_r + a_r \in \mathbb{Z}_{p_r}$ and it is impossible to have $2(i_r + a_r) < p_r$, we must define $c_r = 2(p_r - (i_r + a_r)) - 1$. But $b_r = 2(p_r - a_r) - 1$ implies $c_r = b_r - j_r$.

Subcase 3. Suppose $j_r = 2(p_r - i_r) - 1$ and $b_r = 2a_r$. Then we have $c_r = j_r - b_r$.

Subcase 4. Suppose $j_r = 2(p_r - i_r) - 1$ and $b_r = 2(p_r - a_r) - 1$. Then we have $c_r = j_r + b_r + 2p_r + 1$.

Case 2. Suppose $k = l$. In this case, we simply define $c_0, \ldots, c_l$ according to the Subcases 1–4. Now t is the largest m such that $c_m \neq 0$. However, if no such m exists, then the sum is 0.

Finally, an argument analogous to that for the space complexity of computing $0^B -^B \text{tal}(y)$ shows that the computation $\text{tal}(x) +^B \text{tal}(y)$ can also be carried out within logarithmic space.

Part II. Now we shall do the proof for $B = \text{Bin}(\omega)$. We note that the key to our proof for the case $B = \text{Tal}(\omega)$ is the fact that given a number n in tally on the input tape of a Turing machine, it is possible to explicitly write down $\text{bin}(n)$ and carry out all sorts of binary computations in LOGSPACE on worktapes before reconverting to tally on the output tape. But if the input is a binary number, then it is not at all clear that these computations on the worktapes can be carried out in LOGSPACE. So we shall have to use a different argument to produce our LOGSPACE group with universe $B = \text{Bin}(\omega)$.

In fact, we will use a modified form of the argument from Theorem 3.2.12 to get a LOGSPACE group with universe $\text{Bin}(\omega)$. Recall the LOGSPACE partition $\text{tal}(P) = \bigcup_n \text{tal}(p_n)$ and the function h from the proof of Theorem 3.2.12. Let $Q_n = \{p^k \in Q : p^i \in P_n \text{ for some } i \geqslant k\}$.. Then $Q = \bigcup_n Q_n$ and $Q_m \cap Q_n = \{1\}$ for all $m \neq n$.

Now for each n, define the group $\mathcal{A}_n$ to have universe $\text{Tal}(\omega)$ and operations defined and computed as follows: Given $\text{tal}(n), \text{tal}(x), \text{tal}(y) \in \text{Tal}(\omega)$, first compute a list of the prime powers $p_0 < p_1 < \cdots < p_{k-1}$ below $x + y$ which are in P_n. This can be done within space logarithmic in $\text{tal}(n) + \text{tal}(x) + \text{tal}(y)$ using a modified version of the algorithm PROCEDURE described earler in this proof because we only have to test, for each $p < x + y$, whether p is a prime power and whether $h(\text{tal}(p)) = \text{tal}(2^r(2n + 1))$ for some r. Then find $j_0 < p_0, \ldots, j_k < p_k$ and $b_0 < p_0, \ldots, b_k < p_k$ such that $x = j_0 + j_1 \cdot p_0 + \cdots + j_k \cdot p_0 \cdot p_1 \cdots p_{k-1}$ and $y =$

$b_0 + b_1 \cdot p_0 + \cdots + b_k \cdot p_0 \cdot p_1 \cdots p_{k-1}$. The j_ν's and the b_ν's can be found within logarithmic space, again by using PROCEDURE. As in Case 2 of Part 1 and by the remainder of Part 1, the operations $x +^n y = c_0 + c_1 p_0 + \cdots + c_t p_0 \cdots p_{t-1}$ and $x -^n y = d_0 + d_1 p_0 + \cdots + d_s p_0 \cdots p_{s-1}$ are in LOGSPACE. Thus the sequence $\mathcal{A}_0$, $\mathcal{A}_1, \ldots$ is fully uniformly LOGSPACE over $\mathrm{Tal}(\omega)$ and each component has universe $\mathrm{Tal}(\omega)$. By Lemma 3.3.2 (d), The product $\mathcal{B} = \bigoplus_n \mathcal{A}_n$ is recursively isomorphic to a LOGSPACE group with universe $B = \mathrm{Bin}(\omega)$. Moreover, it follows from the construction that, for each n, the component group $\mathcal{A}_n$ is monic and has spectrum Q_n. And for $i \neq j$, we have $\mathrm{Spec}(\mathcal{A}_i) \cap \mathrm{Spec}(\mathcal{A}_j) = \{1\}$. Hence $\mathrm{Spec}(\mathcal{B}) = Q$.

□

We can now prove our theorem for torsion groups with no upper bound on the orders of the elements as a corollary.

Corollary 3.3.16. *Let Q be an r.e. hereditary locally bounded set with an infinite hereditary subset P such that $\mathrm{tal}(P)$ is in LOGSPACE and $(Q \setminus P) \cup \{1\}$ is hereditary. Then any monic recursive Abelian group $\mathcal{A}$ with $\mathrm{Spec}(\mathcal{B}) = Q$ is recursively isomorphic to a LOGSPACE group $\mathcal{B}$, whose universe B may be taken to be $\mathrm{Tal}(\omega)$ or $\mathrm{Bin}(\omega)$.*

Proof: Fix B to be either $\mathrm{Tal}(\omega)$ or $\mathrm{Bin}(\omega)$. Define recursive sets $M_1 = \{m :$ all prime factors of m are in $P\}$ and $M_2 = \{m :$ no prime factors of m are in $P\}$. Now define the universes of infinite monic recursive subgroups $\mathcal{C}_i$ of $\mathcal{A}$ to be $C_i = \{a \in A : |a|_A \in M_i\}$. Evidently $\mathrm{Spec}(\mathcal{C}_1) = P$ and $\mathrm{Spec}(\mathcal{C}_2) = (Q \setminus P) \cup \{1\}$, and $\mathcal{A}$ is recursively isomorphic to $\mathcal{C}_1 \oplus \mathcal{C}_2$. It follows from the previous theorem that there is a monic LOGSPACE Abelian group $\mathcal{B}_1$ with universe $B_1 = B$ and $\mathrm{Spec}(\mathcal{B}_1) = P$, and it follows from Theorem 3.3.14 (b) that $\mathcal{C}_1$ is recursively isomorphic to $\mathcal{B}_1$. Moreover, by Theorem 3.3.8, we know that $\mathcal{C}_2$ is recursively isomorphic to some LOGSPACE group $\mathcal{B}_2$ with universe $B_2 \subseteq \mathrm{Tal}(\omega)$. It now follows from Lemmas 2.3.14, 3.1.1, and

3.3.2 that the set $\mathcal{B}_1 \oplus \mathcal{B}_2$ is recursively isomorphic to a LOGSPACE group $\mathcal{B}$ with universe B.

□

Theorem 3.3.17. *For any r.e. degree d, there is an infinite r.e. set Q of primes and 1 of degree d such that any monic recursive Abelian group $\mathcal{A}$ with* $\mathrm{Spec}(\mathcal{A}) = Q$ *is recursively isomorphic to a LOGSPACE group $\mathcal{B}$, whose universe B may be taken to be* $\mathrm{Tal}(\omega)$ *or* $\mathrm{Bin}(\omega)$.

Proof: Let $p_0, p_1, \ldots$ enumerate the prime numbers in increasing order. Let D be an r.e. set of degree d, and let $Q = \{p_{2n} : n \in D\} \cup \{p_{2n+1} : n \in \omega\} \cup \{1\}$. Then Q is hereditary and locally bounded, has the same degree as D, and has an infinite subset $P = \{p_{2n+1} : n \in \omega\}$ such that $\mathrm{tal}(P)$ is in LOGSPACE. The result now follows from the previous corollary.

□

The fact that the Turing degree of the set $\mathrm{Spec}(\mathcal{A})$ does not determine whether a recursive torsion Abelian group $\mathcal{A}$ is recursively isomorphic to a LOGSPACE group with standard univers follows from Theorem 4.20 of Cenzer and Remmel [3] which we quote next.

Theorem 3.3.18. *For any r.e. degree d, there is a set M of prime numbers and 1 and of degree d such that no monic recursive torsion Abelian group $\mathcal{G}$ with* $\mathrm{Spec}(\mathcal{G})$ *$= M$ can be isomorphic to any primitive recursive Abelian group with universe either* $\mathrm{Tal}(\omega)$ *or* $\mathrm{Bin}(\omega)$.

We conclude our investigation of LOGSPACE Abelian groups with a negative result about tortsion-free groups which is an immediate consequence of Theorem 4.21 of Cenzer and Remmel [3].

Theorem 3.3.19. *There is a recursive torsion-free Abelian group $\mathcal{G}$ such that $\mathcal{G}$ cannot be recursively embedded in any LOGSPACE Abelian group.*

REFERENCES

[1] T. Baker, J. Gill, and R. Solovay, *Relativisations of the $P = NP$ question*, SIAM J. Comp. 4 (1981), 431-442.

[2] D. Cenzer and J.B. Remmel, *Polynomial-time versus recursive models*, Ann. Pure Appl. Logic 54 (1991), 17-58.

[3] D. Cenzer and J.B. Remmel, *Polynomial-time Abelian groups*, Ann. Pure Appl. Logic 56 (1992), 313-363.

[4] D. Cenzer and J.B. Remmel, *Feasibly categorical Abelian groups*, in Feasible Mathematics II, ed. P. Clote and J. Remmel, Prog. in Comp. Science and Appl. Logic 13, Birkhäuser (1995), 91-154.

[5] D. Cenzer and J.B. Remmel, *Feasibly categorical models*, in Logic and Computational Complexity, ed. D. Leivant, Springer Lecture Notes in Computer Science 960 (1995), 300-312.

[6] D. Cenzer and J.B. Remmel, *Feasible graphs with standard universe*, Ann. Pure Appl. Logic 94 (1998), 21-35.

[7] D. Cenzer and J.B. Remmel, *Complexity and categoricity*, Information and Computation 140 (1998), 2-25.

[8] D. Cenzer and J.B. Remmel, *Complexity-theoretic model theory and algebra*, in Handbook of Recursive Mathematics (Vol. I), ed. Y. Ershov, S.S. Goncharov, A. Nerode and J.B. Remmel, Elsevier Studies in Logic and Found. Math. 138 (1998), 381-513.

[9] J.C.E. Dekker, *Two notes on vector spaces with recursive operations*, Notre Dame Journal of Formal Logic 12 (1971), 329-334.

[10] S. Grigorieff, *Every recursive linear ordering has a copy in DTIME(n)*, J. Symbolic Logic 55 (1990), 260-276.

[11] J. Hartmanis and R.E. Stearns, *On the computational complexity of algorithms*, Trans. Amer. Math. Soc. 117 (1965), 285-306.

[12] J.E. Hopcroft and J.D. Ullman, *Formal Languages and their Relation to Automata*, Addison-Wesley, Reading, Mass. (1969).

[13] A.I. Mal'tsev, *On recursive abelian groups*, Soviet Math.-Doklady 3 (1962), 1431-1434.

[14] G. Metakides and A. Nerode, *Recursion Theory and Algebra*, Lecture Notes in Mathematics 450, Springer-Verlag, Berlin (1975), 209-219.

[15] A. Nerode and J.B. Remmel, *Complexity-theoretic algebra I: Vector spaces over finite fields*, in Structure in Complexity Theory, IEEE Computer Soc. Press (1987), 218-239.

[16] A. Nerode and J.B. Remmel, *Complexity-theoretic algebra II: Boolean algebras*, Ann. Pure Appl. Logic 44 (1989), 71-79.

[17] A. Nerode and J.B. Remmel, *Complexity-theoretic algebra: vector space bases*, in Feasible Mathematics, ed. S. Buss and P.J. Scott, Progr. Comp. Sci. Appl. Logic 9 (1990), 293-319.

[18] A. Nerode and J.B. Remmel, *Polynomial time equivalence types*, in Logic and Computation, ed. W. Sieg, Contemp. Math. 106 (1990), 221-249.

[19] A. Nerode and J.B. Remmel, *Polynomially isolated sets*, in Recursion Theory Week (Oberwolfach 1989), ed. K. Ambos-Spies, G.H. Muller and G.E. Sacks, Springer Lecture Notes in Math. 1432 (1990), 323-362.

[20] J.B. Remmel, *When is every recursive linear ordering of type μ recursively isomorphic to a polynomial time linear ordering over the natural numbers in binary form?*, in Feasible Mathematics, ed. S. Buss and P.J. Scott, Progr. Comp. Sci. Appl. Logic 9 (1990), 321-350.

[21] J.B. Remmel, *Polynomial time categoricity and linear orderings*, in Logical Methods, ed. J.N. Crossley, J.B. Remmel, R.A. Shore and M.E. Sweedler, Progr. Comp. Sci. Appl. Logic 12 (1993), 713-746.

[22] P. Odifreddi, *Classical Recursion Theory*, Elsevier Science Pub. Co., New York (1989).

[23] C. H. Papadimitriou, *Computational Complexity*, Addison-Wesley, Reading, Mass. (1994).

[24] R. Smith, *Two theorems on autostability in p-groups*, Logic Year 1979-80 (Storrs, CT), Lecture Notes in Math. 859, Springer-Verlag, Berlin (1981), 302-311

BIOGRAPHICAL SKETCH

I was born in Bangladesh in 1975. Owing to my father's career in the diplomatic service, I spent 15 of the first 18 years of my life in various countries of the Middle East and then in Germany, where I completed my secondary education. Then I obtained a bachelor's degree in mathematics from the University of Idaho in 1997, and a master's degree in mathematics from the University of Florida in 2000. In 2001, I began studying under Professor Douglas Cenzer and working toward a doctoral degree.

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Douglas A. Cenzer

Douglas A. Cenzer, Chair
Professor of Mathematics

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

[illegible]

William Mitchell
Professor of Mathematics

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

[illegible]

Rick L. Smith
Associate Professor of Mathematics

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

[illegible]

Helmut K. Voelklein
Professor of Mathematics

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

[illegible]

Beverly A. Sanders
Associate Professor of Computer and
Information Science and Engineering

This dissertation was submitted to the Graduate Faculty of the Department of Mathematics in the College of Liberal Arts and Sciences and to the Graduate School and was accepted as partial fulfillment of the requirements for the degree of Doctor of Philosophy.

August 2004

Dean, Graduate School